VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití LIDARu v robotických aplikacích

Exploiting LIDAR in Robotic Applications

2013 Bc. Branislav Holý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Branislav Holý**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití LIDARu v robotických aplikacích
Exploiting LIDAR in Robotic Applications

Zásady pro vypracování:

V dnešní době je kladen vysoký požadavek na bezpečnost osobní dopravy. Tohoto cíle je možno dosáhnou mnoha prostředky. Jednou ze slibných technologií je i LIDAR, který je možno využít pro množství aplikací. Cílem práce je využít LIDARu k detekci a lokalizaci objektů v prostoru, popřípadě k lokalizaci nosiře LIDARového scaneru. Ve své práci proveďte:

1. Nastudujte technologii LIDARu a řádně ji popište.
2. Po konzultaci s vedoucím práce vyberte lokalizační algoritmus.
3. Algoritmus naimplementujete, řádně otestujte a v textu práce popište.
4. V závěru svou práci náležitě zhodnoťte.

Seznam doporučené odborné literatury:

[1] Thrun, S.: Particle Filters in Robotics, Proceedings of the 17th Annual Conference on Uncertainty in AI
[2] Shan, J., Toth, C.: Topographic Laser Ranging and Scanning: Principles and Processing, CRC Press
[3] Popis algoritmů typu SLAM na webové stránce: www.openslam.org

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013

Holý Branislav

.....

Rád bych zde poděkoval panu Ing. Janu Gaurovi za poskytnuté rady a také své rodině, přítelkyni a přátelům, kteří mě při tvorbě této práce podporovali.

Abstrakt

Tato diplomová práce popisuje využití LIDARové technologie pro autonomní mapování prostoru robotem, který je vybavený laserovým dálkoměrem. Dále poukazuje na problém současné lokalizace a mapovaní, který řeší pomocí metody SLAM. K testování těchto technik se využívá robot Neato XV-15, který v neznámé oblasti najde parkovací místo v průběhu autonomního mapování prostoru a za pomocí navigačního algoritmu A* najde k parkovacímu místu cestu. Robot si pamatuje oblast, kterou zmapoval a dokáže tedy dojet zpět na startovací polohu pomocí nejkratší cesty.

Klíčová slova: LIDAR, lokalizace, mapování, SLAM, autonomní mapování, A*

Abstract

This diploma thesis describes exploiting of LIDAR technology for autonomous mapping with robot equipped with laser rangefinder. It also points to the problem of simultaneous localization and mapping, which is solved by SLAM technique. For testing of these techniques is used robot Neato XV-15, which finds parking spot in unknown environment during autonomous mapping of the space. Then the robot finds the shortest path to the parking spot with A* algorithm. The robot remembers explored space and can drive back to the starting position with the shortest path.

Keywords: LIDAR, localization, mapping, SLAM, autonomous mapping, A*

Seznam použitých zkratek a symbolů

LIDAR	–	Light Detection and Ranging
SLAM	–	Simultaneous Localization and Mapping
RANSAC	–	Random Sample Consensus
ICP	–	Iterative Closest Point
SVD	–	Singular Value Decomposition
LCS	–	Longest Common Subsequence

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době, kdy moderní technika pomáhá téměř ve všech činnostech, které člověk vykonává, a v některých jej dokonce plně zastupuje, se v automobilovém průmyslu ještě naplno neprojevila. Určité části automobilů jsou sice ovládány počítačem, ale samotné řízení stále vykonává člověk téměř ve všech případech. V posledních letech však dochází k postupné změně a i samotné řízení začíná vykonávat počítač.

Jedním z prvních aut, která jsou řízená bez účasti člověka, je vozidlo „Stanley". To pod záštitou Googlu a týmu stanfordského profesora Sebastiana Thruna vyhrálo DARPA Grand Challenge v roce 2005, když dokázalo ujet 212 km v Mohavské poušti za 6 hodin a 54 minut bez účasti řidiče [1, 2, 3]. K tomu mu pomáhala GPS anténa, která přinesla přibližnou lokalizaci, kameru a LIDARový 3D skener, díky kterému mohlo vozidlo sestavovat 3D mapu prostředí a lokalizovat se v ní s větší přesností než pomocí GPS [4]. V současné době je „Stanley" umístěn ve Smithsonově národním muzeu americké historie [5]. O dva roky později se stanfordský tým znovu účastnil DARPA Grand Challenge 2007 s automobilem „Junior". V tomto roce byl tým opět úspěšný a skončil na druhém místě [6]. Po těchto vítězstvích se začal Google více věnovat projektu „Driveless car" a v roce 2012 získal oprávnění pro testování jízdy s těmito auty v ulicích měst států Nevada a Florida [7].

Za dalším projektem stojí automobilová společnost Audi, která na letošním veletrhu CES 2013 představila autonomní parkování se svým vozem Audi A7. Demonstrování probíhalo v podzemním parkovišti, kde posádka vozidla vystoupila při vjezdu na parkoviště, a auto si samo našlo parkovací místo, do kterého zaparkovalo. Když posádka chtěla odjet domů, tak automobil sám přijel na místo, kde posádka předtím vystoupila. Audi použilo pro přesnější lokalizaci ultrazvukové senzory, které dokážou zmapovat prostor parkoviště [8].

A právě podobná úloha je cílem této práce. Půjde o nalezení parkovacího místa robotem Neato XV-15 v neznámém prostředí, podobně jako u automobilu Audi A7, avšak za využití LIDARového 2D skeneru, což je technologie, kterou používá pro lokalizaci a mapování Google. Hlavní problém při řešení této úlohy představuje současná lokalizace a mapování. Pokud bychom měli přesnou mapu prostředí a přesný LIDARový skener, tak by úloha lokalizace byla jednoduchá. Stejně tak, kdybychom znali přesnou polohu LIDARového nosiče a měli přesný skener, tak by byla i úloha mapování jednoduchá. Problém nastává v případě, pokud nemáme ani přesný skener, ani mapu prostředí a ani neznáme přesnou polohu LIDARového nosiče. A protože nežijeme v ideálním světě, kde jsou všechny senzory bezchybné, tak musíme tento problém řešit.

Práce je rozdělena do několika částí. Na začátku se seznámíme s LIDARovou technologií a robotem Neato XV-15, který je vybaven LIDARovým skenerem. V následujících dvou kapitolách zjistíme, jakým způsobem můžeme pracovat s daty, které získáme z laserového dálkoměru a odometrie robota, či jiného vozidla. Poté se podíváme na řešení samostatného lokalizování a samostatného mapování a blíže popíšeme problém jejich současného vykonávání. V kapitole číslo 7 se budeme zabývat hlavním problémem současné lokalizace a mapování pomocí techniky SLAM.

V této části již tedy dokáže robot správně zpracovávat data ze skeneru a lokalizovat se v mapě, kterou pomocí skeneru vytváří. Hovoříme o pasivním SLAMu, protože robot ještě nedokáže sám projíždět prostor a vytvářet kompletní mapu. V osmé a deváté kapitole se budeme věnovat tzv. aktivnímu SLAMu, kde robota naučíme plánovat si cestu do cíle a provádět autonomní mapování oblasti. V poslední desáté kapitole se zaměříme na cíl této

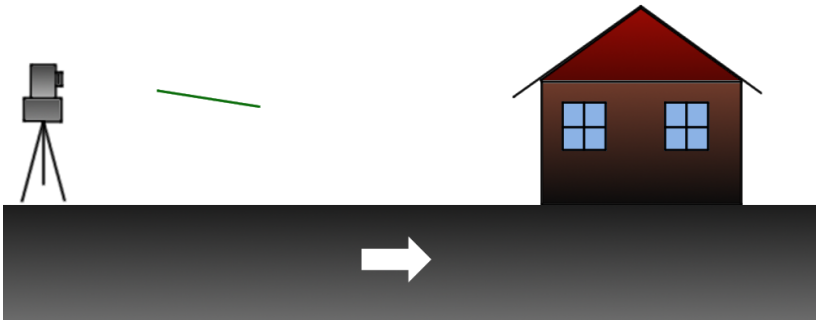
práce a tedy na zaparkování do nalezeného parkovacího místa pomocí výše uvedených technik. Vše následně otestujeme přímo na robotovi Neato XV-15, který by měl v neznámém prostředí najít parkovací místo, zaparkovat do tohoto místa a poté se vrátit zpět na startovací polohu pomocí nejkratší cesty.

2 LIDARová technologie

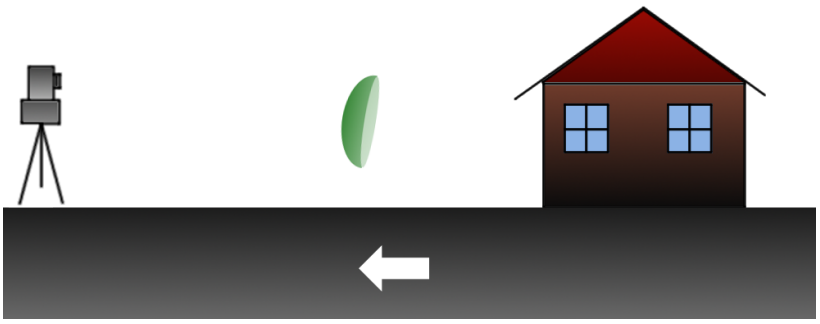
LIDAR je zkratkou pro Light Detection and Ranging, což je metoda, která dokáže změřit vzdálenost ke snímanému objektu nebo jeho jiné vlastnosti. Snímán je vždy cíl, který je nejblíže od LIDARového nosiče ve směru vyslaného laserového paprsku. Informace o cíli se získávají pomocí zpětného rozptylu. Používá se hlavně v archeologii, geografii, geologii, geomorfologii, seismologii, lesnictví, dálkovém průzkumu Země, atmosférické fyzice, leteckém laserovém mapování (ALSM), mapování kontur a dalších oblastech [9].

2.1 Základní princip LIDARové technologie

LIDARový systém se skládá z několika částí. První je laser, který vysílá světelné paprsky o určité vlnové délce a pod určitým úhlem, který se v průběhu skenování mění. Existují 2D skenery, jejichž laser skenuje prostor v určité rovině a také 3D skenery, které snímají trojrozměrný prostor ve třech dimenzích. Další část je optická (buď zrcátko, nebo dělič paprsků), která se stará o směrování laserového paprsku do prostoru. A v poslední části fotodetektor, jenž přijímá zpětný rozptyl (více v podkapitole 2.3), na jehož základě se určí vzdálenost k snímanému předmětu. Na obrázku 1 vidíme, jak LIDARový skener vyslal paprsek pro skenování domu a na obrázku 2 je znázorněno, jak se tento paprsek po dopadu na dům odrazil v podobě zpětného rozptylu.



Obrázek 1: Vyslání LIDARového paprsku



Obrázek 2: Odražený zpětný rozptyl LIDARového paprsku

2.2 Používaná vlnová délka

LIDARové systémy používají různé vlnové délky světelných paprsků, kterými snímají prostor. Od ultrafialových, přes viditelné spektrum až po blízké infračervené záření. Použitá vlnová délka závisí hlavně na prostředí, které chceme LIDARem měřit, abychom získali co možná nejpřesnější údaje.

Vlnové délky mezi 600 - 1000 nm se nejčastěji používají pro nevědecké účely, protože nejsou příliš drahé. Avšak mohou být lehce zachyceny lidským okem a proto je důležité, aby maximální výkon laseru byl omezen kvůli bezpečnosti. Druhou nejčastější možností je 1550 nm. Tato vlnová délka je pro oko bezpečná i při vyšším výkonu, protože není zaznamenána lidským okem. Problémem je ale fotodetektor zpětného rozptylu, který není pro tuto vlnovou délku velmi přesný. Používaná hlavně ve vojenském průmyslu, protože není viditelná skrz brýle pro noční vidění, což pro 1000 nm neplatí. Lasery pro vzdušnou topografii používají většinou vlnovou délku 1064 nm. Hloubkové systémy ovšem používají 532 nm, protože vlnová délka 532 nm proniká vodou s mnohem menším útlumem než 1064 nm a tak je i měření přesnější [9].

2.3 Typy zpětných rozptylů

Vyslaný LIDARový paprsek se odráží od snímaného objektu v podobě zpětného rozptylu (viz obr. 2). Na tento rozptyl se pohlíží z několika pohledů. Nejčastěji jako na Rayleighův rozptyl, Mieův rozptyl, Ramanův rozptyl a fluorescenci [9].

2.3.1 Rayleighův a Mieův rozptyl

Jedná se o rozptyly světla na shlucích molekul plynu. Používá se proto hlavně pro měření v atmosféře. Rayleighův rozptyl je proti Mieově rozptylu omezen dvěma podmínkami:

1. rozptylující částice nemohou být elektricky vodivé,
2. musí platit vztah $\frac{2\pi r}{\lambda} \ll 1$, kde r je poloměr rozptylující částice a λ je vlnová délka světla.

Mieův rozptyl je tedy obecnější a lze tuto teorii použít pro libovolné částice a také pro částice elektricky nabité [10].

2.3.2 Ramanův rozptyl

Ramanův rozptyl se využívá při analýze pevných látek, kapalin a plynů a také při analýze povrchů. Základním principem je dvoufotonový přechod mezi dvěma stacionárními vibračními stavy molekuly, s energiemi E_1 a E_2, vyvolaný vzájemným působením dané molekuly s fotonem dopadajícího světla [11].

2.4 Princip fungování laserového dálkoměru

Pro výpočet vzdálenosti k překážce, kterou snímá LIDARový skener, je zapotřebí čas, který uplynul mezi vysláním laserového paprsku a zaznamenáním odraženého pulzu detektorem. Kvůli velké rychlosti světla není tato metoda vhodná pro přesné měření vzdáleností, kde se vyžaduje přesnost větší než 1 milimetr. Je také možné zjistit, zda se snímaný

objekt pohybuje směrem k LIDARovému nosiči, anebo od něj a také jakou rychlostí, pomocí Dopplerova jevu.

Vzdálenost mezi LIDARovým nosičem a snímaným objektem se vypočítá podle vzorce:

$$D = \frac{ct}{2},$$

kde c je rychlost světla v atmosféře a t je doba, mezi vysláním světelného paprsku a zaznamenáním zpětného rozptylu. Což je:

$$t = \frac{\varphi}{\omega},$$

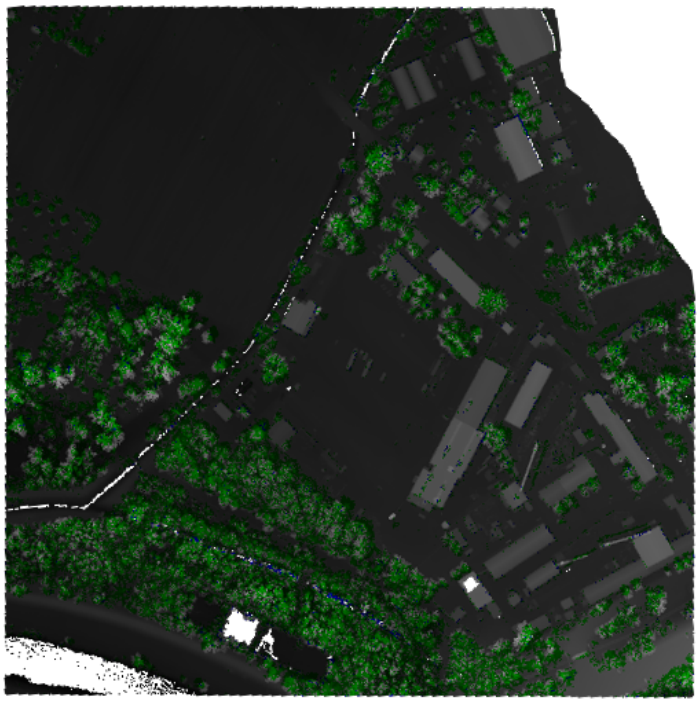
kde φ je fázové zpoždění a ω je úhlová frekvence. Po dosazení vychází:

$$D = \frac{1}{2}ct = \frac{c\varphi}{2\omega} = \frac{c}{4\pi f}(N\pi + \Delta\varphi) = \frac{\lambda}{4}(N + \Delta N),$$

kde λ je vlnová délka paprsku (c/f), $\Delta\varphi$ je část fázového zpoždění ($\varphi \; modulo \; \pi$), N je počet půlvln pro cestu mezi LIDARovým nosičem a snímaným objektem a ΔN je zbývající část půlvln [12].

2.5 Využití LIDARové technologie

LIDARová technologie se využívá v mnoha odvětvích, kde je zapotřebí zmapovat určitou oblast, ať už podzemní, pozemní nebo vzdušnou, či se vyhýbat překážkám při autonomním pohybu. Na obrázku 3 vidíme snímek vytvořený pomocí LIDARových dat v okolí Karviné. Vidíme na něm šedou barvou znázorněné budovy, zelenou stromy a modrou řeku.



Obrázek 3: Snímek LIDARových dat

2.5.1 Archeologie

V oblasti archeologie pomáhá při mapování prvků pod lesní klenbou a poskytuje přehled o hlavních prvcích, které mohou být na zemi k nerozeznání. Používá se také pro vytváření digitálního modelu terénu archeologických nalezišťve vysokém rozlišení. Díky tomu mohou archeologové odhalit mikro reliéf, který je jinak skrytý pod vegetací. Pro analýzu a interpretaci LIDARových dat lze snadno využít geografických informačních systémů [9].

2.5.2 Biologie a ochrana přírody

V biologii a lesnictví lze LIDAR využít pro měření výšky porostu, biomasy a listové plochy. LIDARová technologie dovoluje výzkumníkům měřit nejen výšku porostu, ale určit také biologickou rozmanitost lesů [9].

2.5.3 Fyzika a astronomie

Observatoře po celém světě používají LIDAR k měření vzdálenosti k „odrazkám" umístěných na měsíci, aby mohli změřit polohu měsíce s přesností několika milimetrů. MOLA (zkratka pro Mars Orbiting Laser Altimeter) používal LIDARový systém na družici u Marsu, aby dokázala přesně zmapovat globální topografii Marsu. Ve fyzice se používá LIDAR pro měření hustoty určitých složek střední a vyšší atmosféry, čehož se dá využít pro měření teploty [9].

2.5.4 Geologie a pedologie

Digitální modely terénu ve vysokém rozlišení, vytvořené pomocí vzdušného a pozemního LIDARu, vedly k významným pokrokům v geomorfologii. Schopnost LIDARu rozlišit jemné topografické prvky jako říční terasy a břehy říčních kanálů nebo schopnost změřit vyvýšení zemského povrchu zakrytého lesním porostem, umožnily mnoho nových studií o fyzikálních a chemických procesech, které ovlivňují krajiny [9].

2.5.5 Meteorologie a atmosferické prostředí

První LIDARové systémy byly použity pro studování mraků a složení atmosféry, včetně její struktury. Z počátku, na základě rubínových laserů, byl LIDAR pro meteorologické účely postaven krátce po vynalezení laseru a představuje tak jednu z prvních laserových aplikací v historii [9].

2.5.6 Právní oblast

Policie používá laserové měřiče rychlosti pro měření rychlosti vozidel v silničním provozu. Na základě této informace může rozhodnout, zda došlo k porušení rychlostních limitů [9].

2.5.7 Robotika a autonomní vozidla

U autonomních vozidel nachází LIDAR využití při snímání okolí, aby mohlo vozidlo bezpečně projíždět terénem. Zároveň dochází k budování mapy a rozlišování objektů u zkoumané oblasti, kterou lze vizualizovat a získat tak přehled o terénu a překážkách.

2.5.8 Vojenství

LIDARové systémy s vyšším rozlišením dokážou shromáždit dostatek detailů pro identifikaci cílů, jako jsou například tanky. Další příklad vojenské LIDARové aplikace je ALMDS, což je systém pro leteckou detekci min, který slouží v protiminovém válčení. Další využití přinesl robotický vrtulník v plné velikosti Boeing AH-6, který v červnu roku 2010 provedl plně autonomní let a k detekci překážek použil právě LIDAR [9].

2.5.9 Zemědělství

V zemědělství pomáhá LIDAR určit, pro které oblasti je potřeba použít dražší hnojivo na základě topografické mapy pole. Výzkumníci z Agricultural Research Service dokázali využít LIDARovou technologii pro vytvoření této mapy a rozdělili farmářskou oblast na vysoce, středně a nízko výnosné oblasti [13].

3 Robot Neato XV-15

Robotický vysavač Neato XV-15 je určen k samostatnému vysávání místností. Dokáže si pamatovat místa, která už vysál a pomocí LIDARového skeneru se dokáže vyhnout překážkám. Vysávání ovšem není cílem této práce a tak se u tohoto robota zaměříme hlavně na jeho skener, ze kterého budeme získávat informace o překážkách a pomocí dalšího příkazu mu budeme definovat, jaký pohyb má vykonat.

Obrázek 4: Robot Neato XV-15

3.1 Základní charakteristika

Robot má rozměry 32,5 cm na šířku, 30,5 cm na délku a 10,2 cm na výšku. LIDARovým skenerem tedy snímá prostor v rovině ve výšce 10 cm. Samotný laser používá frekvenci 785 nm [14], což spadá do kategorie blízkému infračervenému záření. Je umístěný v zadní části a je vzdálený 24 cm od přední hrany. Pohybový model odpovídá modelu „tank" (více v kapitole 4) a je tak řízený dvěma kolečky s hřídelí ve vzdálenosti 14,6 cm od přední části robota.

3.2 Popis důležitých příkazů

U robota budeme využívat několika příkazů. Příkazy mohou vracet data, která jsou ve formátu CSV a každý příkaz (at'už vrátil data nebo nikoli) je zakončen znakem control-Z (^Z) [15]. Veškeré příkazy posíláme v podobě výstupního streamu přes vytvořené Wi-Fi spojení a výsledky příkazů přijímáme pomocí vstupního streamu.

3.2.1 Inicializační příkazy

Využívat budeme dva inicializační (resp. ukončovací) příkazy pro přepnutí robota do testovacího módu (TestMode On), ve kterém je schopen vykonávat určité další příkazy a poté příkaz pro zapnutí skeneru (SetLDSRotation On). Stejné příkazy budeme používat i pro vypnutí skeneru a testovacího módu, jen s parametrem Off.

3.2.2 Příkazy pro pohyb

Příkaz SetMotor, který budeme používat pro pohyb robota má následující syntaxi: SetMotor <LWheelDist_Value> <RWheelDist_Value> <Speed_Value>[1], kde <LWheelDist_Value> je vzdálenost, kterou má robot ujet levým kolečkem v milimetrech, <RWheelDist_Value> je vzdálenost pro pravé kolečko v mm a <Speed_Value> je rychlost pohybu v mm/s.

Pokud nastavíme hodnoty vzdáleností, které má robot ujet, pro jednotlivá kolečka, stejná, pak se při kladných hodnotách robot pohybuje přímo dopředu a při záporných hodnotách přímo dozadu (couvá). Pokud nastavíme <LWheelDist_Value> a <RWheelDist_Value> různá, pak se robot pohybuje po kružnici (příp. se otáčí na místě). Výpočet trajektorie pro tento pohyb je uveden podrobně v kapitole 4.

3.2.3 Příkaz pro získání dat ze skeneru

Poslední příkaz, který budeme používat je GetLDSScan. Ten nemá žádné vstupní parametry, ale na výstupu se zobrazí údaje ze skeneru. Ukázka je uvedena ve výpisu 1, kde AngleInDegrees je úhel (ve stupních), ve kterém byl vyslán laserový paprsek. Hodnoty jsou 0 až 359. Dále DistInMM je vzdálenost, kterou LIDARový skener pro daný úhel naměřil (v mm). Intensity je intenzita zpětně odraženého rozptylu a ErrorCodeHEX je chybový kód, kde 0 znamená, že při měření nedošlo k chybě. Hodnota ROTATION_SPEED udává rychlost rotace skeneru v Hz.

Výpis 1: Ukázka dat, získaných ze skeneru

```
1   AngleInDegrees,DistInMM,Intensity,ErrorCodeHEX
2   0,606,330,0
3   1,612,320,0
4   2,617,312,0
5   3,623,323,0
6   4,629,302,0
7   5,636,301,0
8   6,642,307,0
9    ...
10  356,566,364,0
11  357,569,353,0
12  358,571,352,0
13  359,575,382,0
14  ROTATION_SPEED,4.91
```

[1] U příkazu SetMotor uvádíme pouze některé parametry.

4 Odometrie

Odometrie je proces, díky kterému můžeme zjistit polohu a orientaci robota, resp. jejich změnu ($\Delta x, \Delta y$ a $\Delta\alpha$) na základě informací získaných z pohybového modulu robota, případně z příkazů, které jsme robotovi pro pohyb určili. Využívá se při tom určitých geometrických znalostí, podle typu robota, u kterého odometrii řešíme. V této kapitole popíšeme výpočet trajektorie pro robota Neato XV-15.

4.1 Typy robotů

Mezi nejběžnější typy robotů patří typ auto, všesměrový robot a typ tank [16]. Tyto typy se liší umístěním koleček, které slouží k pohybu robota. Kvůli těmto odlišnostem se trajektorie pro jednotlivé typy počítají různým způsobem.

Typ auto se ovládá pomocí natočení předních kol, čímž se určí směr, kterým se bude robot pohybovat. Tento typ robota se nemůže otáčet na místě a může se pohybovat pouze dopředu nebo dozadu. Označuje se také jako „Ackermanovo řízení" a je použit v dnešních automobilech.

U všesměrových robotů nezáleží na aktuální orientaci robota a dokážou se pohybovat všemi směry. Má tři kolečka, která jsou symetricky rozmístěná kolem osy robota a umožňují libovolný pohyb. Nejedná se o příliš rozšířený typ robota, ale v poslední době se začíná prosazovat v robotickém fotbale.

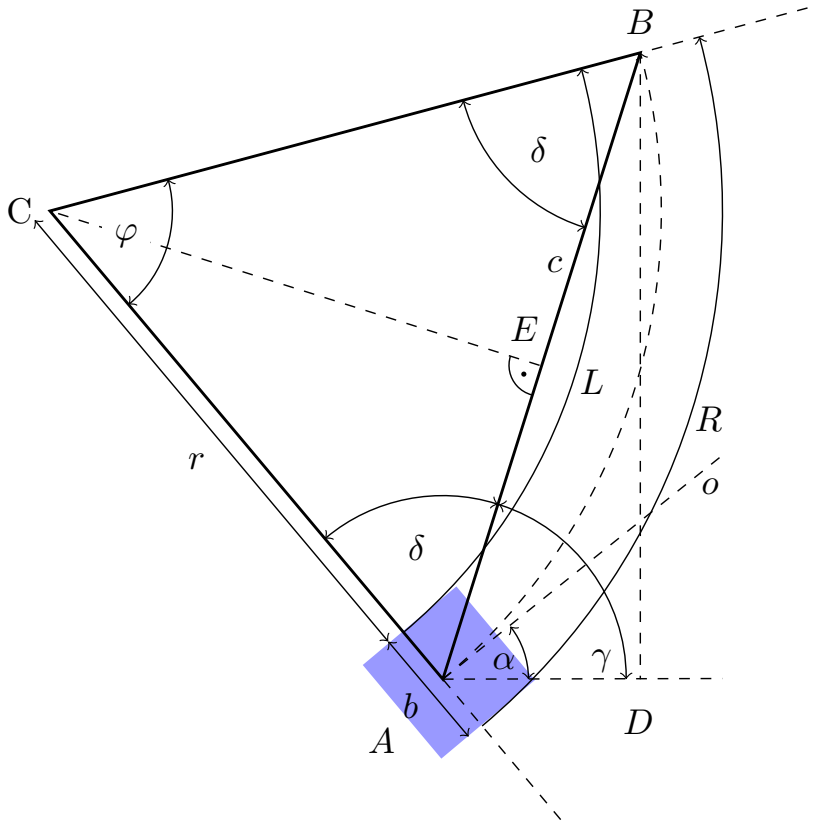
Roboti typu tank jsou řízení pomocí dvou koleček umístěných na kraji robota. Robot se dokáže otáčet na místě a to tehdy, když se kolečka pohybují stejnou rychlostí, ale v opačném směru. Pokud se kolečka pohybují stejnou rychlostí ve stejném směru, tak se robot pohybuje buď dopředu, nebo dozadu. Dále se může robot pohybovat po kružnici a to tehdy, když mají kolečka jinou rychlost. Robot Neato XV-15 je robotem typu „tank" a tak se v následujících podkapitolách budeme zabývat řešením trajektorie a určováním polohy pro tento typ robota.

4.2 Určení trajektorie

U robota typu tank je obecnou trajektorií kružnice. Tu opisují kolečka robota nebo střed nápravy, který se považuje za bod pro lokalizaci robota v mapě. Kružnice může mít určité vlastnosti, podle toho, jaký pohyb robot vykonává. Druhy pohybů jsou: přímý pohyb, kdy robot jede dopředu nebo dozadu po přímce, druhým druhem je otáčení na místě a třetím je pohyb po kružnici.

U prvního druhu pohybu je, podle intuice, trajektorií přímka. Tu si ovšem můžeme také představit jako kružnici s nekonečným poloměrem. Taková kružnice bude mít nulové zakřivení a odpovídá tedy přímce. Při druhém druhu pohybu se robot otáčí na místě kolem své osy. Můžeme si u tohoto případu představit, že trajektorie je bod, což představuje kružnici s nulovým poloměrem. Třetí druh je nejobecnější, kdy má kružnice konečný poloměr a její střed je umístěn mimo robota. Pokud vyřešíme dráhu trajektorie pro tento obecný druh pohybu, tak bude platit i pro zbylé dva.

Následující obrázek 5 zobrazuje vše, co budeme pro určení nové polohy a orientace robota potřebovat. Robotovi posíláme hodnoty pro levé a pravé kolečko, kde určujeme, jakou vzdálenost má robot těmito kolečky urazit. Na obrázku je robot (modrý čtverec) na poloze bodu A a když levé kolečko urazí vzdálenost L a pravé vzdálenost R, tak se robot dostane na polohu B.



Obrázek 5: Geometrie pro výpočet odometrie

4.2.1 Výpočet translace

Rozdíl souřadnic x a y bodů A a B nám určuje naše hledané hodnoty Δx a Δy. Ty vypočítáme z pravoúhlého trojúhelníku ABD jako:

$$\Delta x = c \cos\gamma,$$

$$\Delta y = c \sin\gamma.$$

Úhel γ určíme jednoduše z pohledu na okolí bodů A. Vidíme zde, že osa robota o je kolmá k hřídeli (hřídel leží na stejné přímce jako body A a C) a tedy platí:

$$\gamma = \alpha + \frac{\pi}{2} - \delta,$$

kde úhel δ vypočítáme z trojúhelníku ABC jako:

$$2\delta + \varphi = \pi,$$

$$\delta = \frac{\pi}{2} - \frac{\varphi}{2}.$$

Po dosazení do vzorce pro výpočet úhlu γ dostáváme:

$$\gamma = \alpha + \frac{\pi}{2} - \left(\frac{\pi}{2} - \frac{\varphi}{2}\right),$$

$$\gamma = \alpha + \frac{\varphi}{2},$$

kde úhel α je počáteční orientace robota a úhel φ vypočítáme podle následující soustavy rovnic.

Délka kruhového oblouku se vypočítá jako $l = r\theta$, kde l je hledaná délka, r je poloměr kružnice a θ je úhel kruhového oblouku. V našem případě délka trajektorie pro levé a pravé kolečko se vypočítá jako:

$$\begin{aligned} L &= r\varphi \\ R &= (r+b)\,\varphi. \end{aligned}$$

Když tuto soustavu rovnic vyřešíme, tak získáme úhel φ jako:

$$\varphi = \frac{R-L}{b},$$

kde R je délka trajektorie pravého kolečka, L je délka trajektorie levého kolečka a b je rozteč mezi kolečky.

Po dosazení do vzorce pro výpočet úhlu γ získáme vztah, kde známe všechny hodnoty a můžeme tedy úhel γ vypočítat podle:

$$\gamma = \alpha + \frac{R-L}{2b}.$$

Dále pro výpočet Δx a Δy potřebujeme znát délku strany c. Její polovinu můžeme vypočítat z pravoúhlého trojúhelníku ACE jako:

$$\begin{aligned} \sin\left(\frac{\varphi}{2}\right) &= \frac{\frac{c}{2}}{r+\frac{b}{2}} \\ \frac{c}{2} &= \left(r+\frac{b}{2}\right)\sin\left(\frac{\varphi}{2}\right) \\ c &= 2\left(r+\frac{b}{2}\right)\sin\left(\frac{\varphi}{2}\right), \end{aligned}$$

kde r získáme z výše uvedené soustavy rovnic pro výpočet kruhového oblouku jako:

$$r = \frac{bL}{R-L}$$

a po dosazení r a φ do vzorce pro výpočet délky strany c nám vyjde:

$$\begin{aligned} c &= 2\left(\frac{bL}{R-L}+\frac{b}{2}\right)\sin\left(\frac{R-L}{2b}\right) \\ c &= \frac{b\,(L+R)}{R-L}\sin\left(\frac{R-L}{2b}\right), \end{aligned}$$

kde opět známe všechny hodnoty a můžeme tak vzdálenost c mezi body A a B vypočítat. Následně nám již nic nebrání k výpočtu hodnot Δx a Δy pomocí délky c a úhlu γ, jak je uvedeno na začátku této podkapitoly.

4.2.2 Výpočet rotace

Vzhledem k tomu, že se orientace hřídele změní při pohybu z bodu A do bodu B, o úhel φ (jak lze vidět pomocí trojúhelníku ABC), tak se i orientace robota změní o tento úhel. A tedy platí, že: $\Delta\alpha = \varphi$.

4.2.3 Pohled na speciální druhy pohybu

Na začátku kapitoly jsme rozdělili druhy pohybů robota typu „tank" na tři druhy. Odvodili jsme výpočet pro zjištění Δx, Δy a $\Delta\alpha$ pro obecnou trajektorii kružnice, čili pro třetí druh. Nyní se podíváme, zda skutečně stačí tento výpočet i pro pohyb po přímce a pro rotaci na místě.

Pro pohyb po přímce jsme řekli, že se jedná o pohyb po kružnici s nekonečně velkým poloměrem. Tento poloměr r_t lze vypočítat jako:

$$\begin{aligned} r_t &= r + \frac{b}{2} \\ r_t &= \frac{bL}{R-L} + \frac{b}{2} \\ r_t &= \frac{b\,(L+R)}{2\,(R-L)}, \end{aligned}$$

pokud robotovi pošleme stejné hodnoty pro R a L. Pak robot jede po přímce a při výpočtu poloměru dochází k dělení nulou a poloměr tedy odpovídá ∞, což odpovídá původní hypotéze. Tento poloměr používáme pro výpočet vzdálenosti mezi body A a B, což je na obrázku 5 znázorněno jako hodnota c. Tato hodnota, při pohybu po přímce, odpovídá hodnotě L, resp. R. Pro přesnější určení pak:

$$c = \frac{L+R}{2}.$$

Tato jediná úprava stačí, pro výpočet Δx, Δy a $\Delta\alpha$, pokud robotovi pošleme stejné hodnoty L a R.

Dalším speciálním druhem pohybu je rotace na místě. K té dochází, když robotovi pošleme hodnoty R a L takové, že platí: $L = -R$. Na začátku kapitoly jsme si řekli, že u tohoto druhu pohybu má kružnicová trajektorie nulový poloměr. Pokud do vzorce pro výpočet r_t dosadíme $L = -R$, pak $r_t = 0$, což odpovídá původní domněnce o nulovém průměru kružnice. Díky tomu nám také délka trajektorie c vyjde nula, což znamená, že robot zůstane na místě, a tedy $\Delta x = 0$ a $\Delta y = 0$. Může se změnit jen orientace robota $\Delta\alpha$, která se vypočítá stejně, jako jsme uvedli v podkapitole 4.2.2.

4.3 Určení polohy na trajektorii v čase

V průběhu pohybu robota potřebujeme určit, kde se robot aktuálně nachází, když jsme mu před určitou dobou odeslali příkaz pro vykonání pohybu. V předchozí podkapitole jsme vypočítali, jaké budou nové souřadnice a orientace robota, až svůj pohyb dokončí, čili, jaké jsou souřadnice bodu B a jaká je orientace robota v tomto bodě. Nyní se podíváme na určení polohy a orientace v jakémkoli bodě trajektorie, resp. v jakékoli době mezi začátkem a koncem pohybu.

Pokud potřebujeme zjistit bod na trajektorii pohybu robota, tak nám stačí, když hodnoty L a R vynásobíme procentuálním poměrem, který nás zajímá. Pro zjištění, kde se robot nachází ve dvou třetinách své cesty, je $L' = \frac{2}{3}L$ a $R' = \frac{2}{3}R$, kde L' označuje novou hodnotu, kterou dosadíme do vzorce pro výpočet polohy a orientace, a L určuje hodnotu, kterou jsme robotovi poslali pro levé kolečko. Stejně tak je tomu i u R' a R, kde jsou hodnoty, které se týkají pravého kolečka.

Jestli nás zajímá poloha robota po uplynutí určitého času, tak musíme nejdříve zjistit, jak dlouho se bude robot pohybovat a poté určit procentuální poměr. To ovšem závisí na způsobu, jakým robot ovládá kolečka při příkazu pro pohyb. Zaměříme se proto pouze na robota Neato XV-15.

4.3.1 Určení polohy na trajektorii v čase pro robota Neato XV-15

Robot Neato XV-15 používá příkaz SetMotor (více v podkapitole 3.2.2), ve kterém se určuje rychlost pohybu Speed_Value pro kolečko s delší trajektorií. Aby se kolečka točila stejně dlouhou dobu, tak se kolečko s kratší trajektorií točí pomaleji. Výsledná doba t_f, která je potřebná pro pohyb z bodu A do bodu B, je pak dána jako:

$$t_f = \frac{\max(L, R)}{Speed\_Value},$$

kde funkce $\max$ vrátí číslo L, pokud $L > R$ nebo vrátí R, pokud $L < R$. Pokud jsme poslali robotovi příkaz SetMotor v čase t_0, tak v čase t_1 se robot nachází na poloze vypočítanou pomocí hodnot L' a R', kde:

$$L' = \frac{t_1 - t_0}{t_f} L,$$
$$R' = \frac{t_1 - t_0}{t_f} R.$$

5 Vizualizace a analýza LIDARových dat

LIDARová data nám poskytují informace o prostředí, kde byla naskenována. Tyto informace jsou například u laserového dálkoměru, kterým disponuje i robot Neato XV-15, dostupná jako sekvence délek k nejbližším překážkám (viz výpis 1). Zhlédnutí samotných čísel nám však neukáže, jak dané prostředí vypadá a proto je třeba tyto údaje zobrazit do podoby mapy.

Tato data můžeme analyzovat a určit vlastnosti, které naskenovaná oblast má. Vzhledem k tomu, že řešíme skenování vnitřního prostředí, tak se zaměříme na detekci přímek, resp. úseček, protože vnitřní prostředí je zpravidla tvořeno rovnými zdmi. Tyto zdi se protínají v rozích, které budeme následně v datech detekovat a zpřesňovat jejich polohu výpočtem.

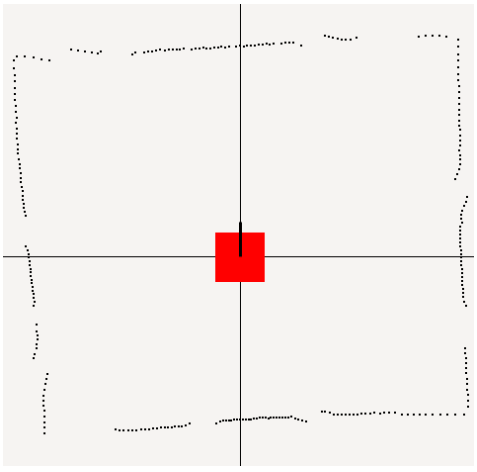
5.1 Vizualizace LIDARových dat

Pro zobrazení LIDARových dat z dálkoměru potřebujeme vypočítat polohu bodu, kde narazil laserový paprsek na nejbližší překážku. Souřadnice x a y tohoto bodu jsou určeny vždy relativně vůči poloze a orientaci robota, resp. LIDARového nosiče. Vstupem pro výpočet polohy tohoto bodu je výstup laserového dálkoměru, který nám poskytuje vzdálenost k nejbližší překážce a úhel, pod kterým byl daný paprsek vyslán. Z těchto dvou údajů, spolu se znalostí polohy a orientace robota, můžeme vypočítat souřadnice x a y tohoto bodu v mapě podle:

$$x = x_s + d\ \cos(\alpha + \theta)$$
$$y = y_s + d\ \sin(\alpha + \theta),$$

kde x_s a y_s jsou souřadnice polohy LIDARového skeneru v mapě, α je orientace skeneru v mapě, d je vzdálenost, kterou skener naměřil a θ je úhel, pod kterým byl paprsek vyslán.

Na obrázku 6 je zobrazeno, jak robot Neato VX-15 naskenoval kolem sebe čtyři rovné desky o délce přibližně 80 cm. Vidíme, že na některých místech došlo k chybě měření, kde body nejsou vůbec zobrazeny. Další chyba je patrná v místech, kde se body odchylují od přímky, kterou by v ideálním případě měly zobrazovat.



Obrázek 6: Vizualizace LIDARových dat

5.2 Detekce přímek a úseček algoritmem RANSAC

Jak již bylo výše napsáno, LIDARový systém nám poskytuje údaje, na jejichž základě si můžeme vytvořit kolekci A s body $A_0, A_1, A_2, \ldots, A_n$, které zobrazují nejbližší místa v okolí. Ve vnitřním prostředí budou tato data určovat body, které leží na přímce, pokud bude robot snímat rovnou plochu, což může být zeď, dveře, skříň a podobně. Bohužel nemůžeme tyto body postupně spojit a vytvořit tak přímku, protože body leží pouze v okolí určité přímky, kvůli chybě měření, kterou LIDARový skener způsobuje. Algoritmus RANSAC [17] nám dokáže najít tuto přímku, resp. vybere z kolekce A takové body, které se nejvíce blíží nějaké přímce, a následně pomocí metody nejmenších čtverců určíme přímo přímku, která těmto bodům odpovídá.

RANSAC se provádí iterativně pro předem stanovený počet iterací I, kde se v každé z nich nejprve náhodně vybere k bodů a vytvoří se z nich model, který určuje, jakou křivku chceme z dat vytvořit. Nás zajímá přímka a tak nám stačí zvolit $k = 2$ a mezi těmito body vytvořit přímku pomocí obecné rovnice přímky $ax + by + c = 0$. Nestačí nám směrnicová rovnice přímky, kterou algoritmus RANSAC často používá, protože může dojít k situaci, kdy jsou vybrány body, které leží na vertikální přímce.

Následně spočítáme chybu, kterou aktuálně vybraný model má. Ta se určí jako součet čtverců vzdáleností od všech bodů k našemu modelu. Zároveň máme v algoritmu určenou maximální možnou vzdálenost bodu od přímky $dist_{max}$ a pokud je překročena, tak se přičte pouze druhá mocnina této maximální vzdálenosti.

Vzdálenost $v_{i,j}$ bodu $A_j = [x_j, y_j]$ od přímky i-tého modelu $p_i : a_i x + b_i y + c_i = 0$ je dána podle:

$$v_{i,j} = \frac{|a_i x_j + b_i y_j + c_i|}{\sqrt{a_i^2 + b_i^2}}.$$

Při přidání podmínky pro maximální vzdálenost a výpočet čtverce vzdáleností, pak:

$$v_{i,j}^2 = \begin{cases} \frac{(a_i x_j + b_i y_j + c_i)^2}{a_i^2 + b_i^2} & \text{pro } v_{i,j} < dist_{max} \\ dist_{max}^2 & \text{pro } v_{i,j} \geq dist_{max} \end{cases}$$

Samotná suma vzdáleností pro i-tý model je tedy dána jako:

$$err_i = \sum_{j=0}^{n} v_{i,j}^2,$$

kde n určuje počet bodů, mezi kterými přímku hledáme. Při počítání hodnoty err_i si zároveň ukládáme takové body, které při výpočtu vzdálenosti splnily podmínku $v_{i,j} < dist_{max}$. Tyto body poté tvoří kolekci M o délce m.

Cílem je zjistit, která přímka má nejmenší hodnotu err_i a uložené body kolekce M, pro tento model, jsou poté použity pro konečný výpočet parametrů přímky, která se nejvíce přibližuje vybraným bodům. Po provedení I iterací tedy získáme jednu kolekci M. Abychom mohli v datech najít více přímek, tak stačí, když z původní kolekce A odebereme body v kolekci M a algoritmus provedeme znovu.

5.2.1 Metoda nejmenších čtverců

Tato metoda [18] dokáže najít parametry křivky, která se nejvíce přibližuje určeným bodům. Při detekci přímek máme pomocí algoritmu RANSAC vybrané body v kolekci M, které leží v okolí nějaké přímky. Nyní budeme používat směrnicový tvar přímky $y = kx + q$, který nakonec převedeme na obecný tvar přímky a výpočet opatříme podmínkou, určenou pro body ležící na vertikální přímce.

Pro směrnicový tvar přímky tedy hledáme takové koeficienty k a q, kde součet čtverců vzdáleností je minimální. Vzdálenost bodu od přímky můžeme určit jako rozdíl y souřadnic počítaného bodu a bodu, který má stejnou x-ovou souřadnici, ale leží na dané přímce. A tedy čtverec vzdálenosti Δy^2 má tvar:

$$\Delta y_i^2 = (kx_i + q - y_i)^2,$$

kde x_i a y_i jsou souřadnice počítaného bodu a k a q jsou parametry dané přímky. Jde nám tedy o nalezení minima funkce $S(k, q)$, která je definována právě jako suma čtverců vzdáleností pro všechny body v kolekci M.

$$S(k,q) = \sum_{i=0}^{m} \Delta y_i^2 = \sum_{i=0}^{m} (kx_i + q - y_i)^2$$

Minimum funkce můžeme vypočítat, když položíme parciální derivace rovny nule. Při derivaci podle k a q dostáváme:

$$\frac{\partial S}{\partial k} = \sum_{i=0}^{m} 2x_i(kx_i + q - y_i) = 2\left(k\sum_{i=0}^{m} x_i^2 + q\sum_{i=0}^{m} x_i - \sum_{i=0}^{m} x_i y_i\right)$$

$$\frac{\partial S}{\partial q} = \sum_{i=0}^{m} 2(kx_i + q - y_i) = 2\left(k\sum_{i=0}^{m} x_i + q\sum_{i=0}^{m} 1 - \sum_{i=0}^{m} y_i\right) = 2\left(k\sum_{i=0}^{m} x_i + qm - \sum_{i=0}^{m} y_i\right).$$

Když tyto parciální derivace dáme rovny nuly, tak získáme soustavu dvou rovnic o dvou neznámých:

$$\begin{aligned} k\sum_{i=0}^{m} x_i^2 + q\sum_{i=0}^{m} x_i &= \sum_{i=0}^{m} x_i y_i \\ k\sum_{i=0}^{m} x_i + qm &= \sum_{i=0}^{m} y_i. \end{aligned}$$

Koeficienty k a q pro přímku, ke které jsou všechny body nejblíže, pak získáme po vyřešení soustavy rovnic jako:

$$k = \frac{m\sum\limits_{i=0}^{m} x_i y_i - \sum\limits_{i=0}^{m} x_i \sum\limits_{i=0}^{m} y_i}{m\sum\limits_{i=0}^{m} x_i^2 - \sum\limits_{i=0}^{m} x_i \sum\limits_{i=0}^{m} x_i}$$

$$q = \frac{\sum\limits_{i=0}^{m} y_i \sum\limits_{i=0}^{m} x_i^2 - \sum\limits_{i=0}^{m} x_i \sum\limits_{i=0}^{m} x_i y_i}{m\sum\limits_{i=0}^{m} x_i^2 - \sum\limits_{i=0}^{m} x_i \sum\limits_{i=0}^{m} x_i}.$$

Nyní můžeme převést směrnicový tvar přímky s koeficienty k a q na obecný tvar s koeficienty a, b a c, kde:

$$a = k,$$

$$b = -1,$$

$$c = q.$$

V případě, kdy platí následující vztah:

$$m \sum_{i=0}^{m} x_i^2 - \sum_{i=0}^{m} x_i \sum_{i=0}^{m} x_i = 0,$$

pak vychází přímka vertikální a není možné vypočítat koeficienty k a q, protože by při výpočtu docházelo k dělení nulou. V takovém případě můžeme hodnoty a, b a c určit přímo jako:

$$a = 1,$$

$$b = 0,$$

$$c = -\frac{\sum_{i=0}^{m} x_i}{m}.$$

Po provedení dalších iterací algoritmu RANSAC a následném výpočtu koeficientů pro všechny přímky, pak získáme mapu prostředí jako na obrázku 7. Zde černé přímky označují osy x a y v mapě, černé body představují naměřené hodnoty z LIDARového skeneru a přímky jiných barev pak vizualizují nalezené přímky pomocí tohoto algoritmu.

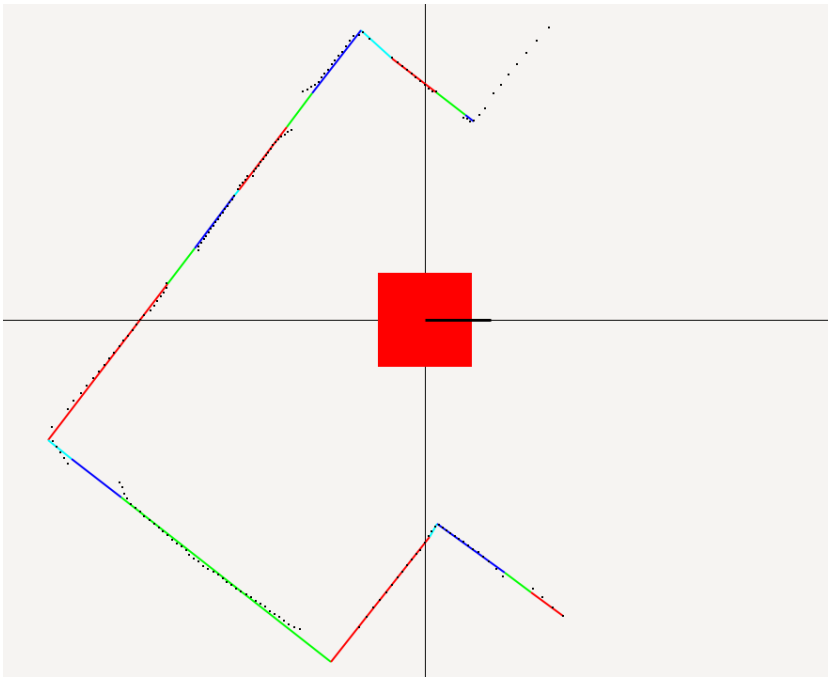
Obrázek 7: Detekce přímek algoritmem RANSAC

5.2.2 Rozdělení přímek na úsečky

Přímky nalezené algoritmem RANSAC, jak jsou zobrazeny na obrázku 7, přirozeně zasahují do míst, kde se nenachází žádné překážky. Z tohoto důvodu se nedají použít pro určení hranice překážek a zdí. Tyto přímky musíme omezit pouze na oblast, ve které jsou také body, které LIDARový skener naměřil.

Vzhledem k tomu, že známe body kolekce M, tj. $M_0, M_1, M_2, \ldots, M_m$, které patří každé přímce, a také známe pořadí bodů, podle úhlu ve kterém byly naskenovány, tak můžeme tyto body pro každou přímku seřadit a postupně procházet. Při procházení zjišťujeme, zda dva sousední body byly opravdu naskenované po sobě, pokud ano, tak patří jedné úsečce. Když najdeme místo, kde dva po sobě jdoucí body nebyly naskenovány po sobě, tak přestaneme s tvorbou aktuální úsečky a začneme tvořit další úsečku. Tím rozdělíme body kolekce M do několika skupin $S_0, S_1, S_2, \ldots, S_s$, kde $S_0 \cup S_1 \cup S_2 \cup \cdots \cup S_s = M$ a kde každá skupina S_i představuje body jedné úsečky. Samotnou úsečku resp. její krajní body pak vypočítáme, když zjistíme průsečíky dané přímky s kolmicemi, které procházejí prvním a posledním bodem ze skupiny S_i.

Po provedení těchto kroků získáme s úseček, které nejsou spojené. Pokud je poslední bod jedné úsečky blízko prvního bodu druhé úsečky, tak můžeme vytvořit další úsečku, kterou tyto dvě úsečky spojíme. Detail výsledku rozdělení přímek z obrázku 7 je zobrazen na obrázku 8, kde vidíme, jednotlivé úsečky zobrazeny jinou barvou. Zároveň tyto úsečky kopírují zeď, kterou skener naskenoval a tak již nezasahují do prostoru bez překážek.



Obrázek 8: Rozdělení přímek z algoritmu RANSAC na úsečky

5.3 Detekce význačných bodů a rohů

První z analyzovaných vlastností prostředí byly přímky, resp. úsečky, které můžeme použít pro znázornění překážek. Další důležitou vlastností, která se využívá pro lokalizování, jsou význačné body. Tyto body v prostoru mají jednu důležitou vlastnost, a sice že můžeme ve skenování najít význačné body z předchozích skenování a propojit tak navzájem korespondující význačné body v rámci několika skenování.

Jaké body, či oblasti v prostoru to budou, závisí na prostředí, ve kterém se robot pohybuje. Při průjezdu parkem s několika stromy můžeme za význačné body považovat středy kmenů stromů, které naskenujeme. Ve vnitřním prostředí to pak mohou být rohy, které typicky svírají zdi pokojů, nábytek se zdí a podobně.

Protože řešíme pohyb robota ve vnitřním prostředí, tak se zaměříme právě na rohy. V další části této podkapitoly si ukážeme, jak zpřesnit polohy rohů výpočtem. Díky tomu bude docházet k menším chybám při lokalizaci, protože budeme vycházet z přesnějších dat.

5.3.1 Nalezení rohů

Pod rohem si představme bod v prostoru, kde dvě stěny svírají vnitřní úhel menší než úhel β. Pro výpočet úhlu mezi stěnami nám pomůže předchozí detekce úseček. Předpokládejme tedy, že máme uloženou kolekci úseček, které jsme vypočítali podle kapitoly 5.2 a kde každá úsečka je definována svým počátečním a koncovým bodem. Tyto dva body sice nejsou jedny z těch, které nám poskytl LIDARový skener, což ovšem vůbec nevadí, protože při detekci rohů jde o nalezení bodů, které se nejvíce přibližují místům s rohy v prostředí, a nejde o určení, které z naskenovaných bodů jsou význačnými body, tedy rohy. Je totiž velmi pravděpodobné, že mezi naskenovanými body nebude žádný bod, který by přesně odpovídal rohu.

Vzhledem k tomu, že každá úsečka má také uloženy body ze skeneru, tak můžeme tyto úsečky seřadit, protože víme, jaké je pořadí jednotlivých naskenovaných bodů. Po seřazení můžeme úsečky procházet a každou sousedící dvojici převést na dva vektory a a b, které vycházejí ze společného bodu, kde se úsečky dotýkají. Poté můžeme vypočítat vnitřní úhel α, který vektory a a b svírají podle skalárního součinu:

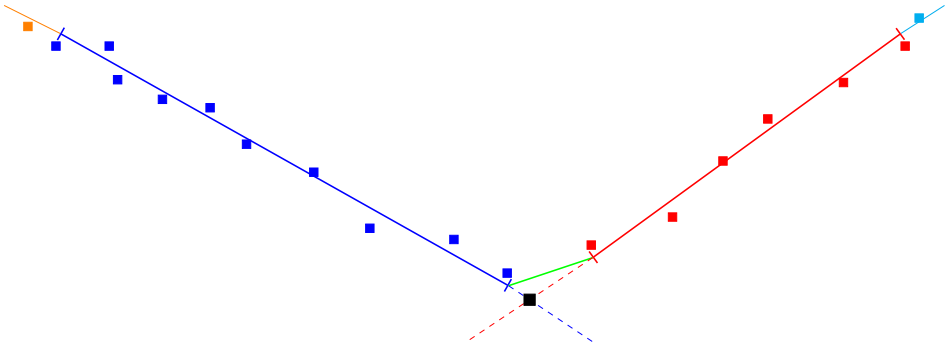
$$\alpha = \arccos\left(\frac{a \cdot b}{|a||b|}\right).$$

Pokud pro tyto dva vektory platí $\alpha < \beta$, tak si koncový bod první úsečky, resp. první bod druhé úsečky přidáme do kolekce pro význačné body.

5.3.2 Vypočítané rohy

Častokrát ovšem dojde k situaci, kdy dvě delší úsečky, které představují části zdí kolem rohu, mají mezi sebou další úsečku, která vznikla pouze spojením těchto dvou úseček. Situace je zobrazena na obrázku 9. Zde vidíme modrou a červenou úsečku znázorňující zdi a oblast kolem rohu je spojená zelenou úsečkou.

Při použití výše zmíněného algoritmu pro detekci rohů bychom za roh považovali nejspíše bod, kde se spojuje modrá a zelená úsečka, protože svírají ostřejší úhel než úsečka červená a zelená. Mohli by ovšem také dojít k takové situaci, kdy by k detekci rohu vůbec nedošlo. Příčinou by bylo, že by obě zdi se spojovací zelenou úsečkou nesvíraly dostatečně ostrý úhel, ale zdi mezi sebou by dostatečně ostrý úhel měly.



Obrázek 9: Zobrazení rohu pro výpočet přesného rohu

Pro tyto situace je výhodnější, když roh dopočítáme. To můžeme udělat, když při procházení úseček narazíme na takovou i-tou úsečku, která neobsahuje žádné body ze skeneru (na obrázku 9 nemá zelená úsečka žádné zelené čtverečky) a její sousední úsečky, čili ty na pozicích $i+1$ a $i-1$, mezi sebou svírají úhel α, kde $\alpha < \beta$. V tomto případě můžeme vypočítat průsečík (černý čtvereček na obrázku 9) mezi přímkami, které proložíme těmito úsečkami. Obě úsečky (červenou i modrou) pak můžeme protáhnout až do tohoto průsečíku, přičemž se stane zelená úsečka zbytečná a tak ji můžeme z kolekce odstranit.

Tímto algoritmem jsme si zpřesnili jednak polohu rohu v mapě, ale také jsme zpřesnili hranici zdí, která teď více odpovídá skutečnosti. V následujících kapitolách budeme význačné body používat pro lokalizaci a pro současnou lokalizaci a mapování. U těchto technik závisí úspěšný výsledek na přesnosti skeneru, kterou jsme nyní pro detekci rohů zlepšili.

6 Lokalizace a mapování

Problém samostatné lokalizace můžeme řešit za předpokladu, že známe mapu prostředí, ve které se robot pohybuje. Poté, na základě měření ze skeneru, můžeme s určitou pravděpodobností určit, ve kterých oblastech mapy se robot nejpravděpodobněji nachází. Pro samostatné mapování potřebujeme znát přesnou polohu robota, resp. LIDARového nosiče, v době provádění skenování, abychom mohli tyto naskenované snímky správně poskládat do celistvé mapy. Pokud bychom chtěli vykonávat tyto dvě činnosti současně, neměli bychom tedy k dispozici mapu prostředí a ani bychom nemohli znát přesnou polohu robota, tak zde vidíme paradox. Pro výpočet lokalizace potřebujeme mapu, kterou ovšem můžeme zjistit pouze, pokud známe lokalizaci.

6.1 Lokalizace

Pro řešení lokalizace se využívá hlavní smyčky, která vykonává dvě základní operace a těmi jsou *skenování* a *pohyb*. Díky skenování získáváme informace o poloze robota a při každém pohybu tuto informaci ztrácíme. Nyní si ukážeme 2 techniky pro lokalizaci a těmi jsou Monte Carlo lokalizace a Particle filtr.

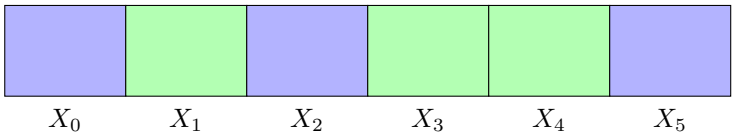
6.1.1 Monte Carlo lokalizace

Lokalizace Monte Carlo [19], známá též jako Histogram filtr, je jednou z metod lokalizace pro diskrétní prostory. Využívá multimodálního rozložení pravděpodobnosti a tak může mít více oblastí v mapě stejnou pravděpodobnost výskytu robota ve stejném čase. Výsledkem je určení pravděpodobností výskytu robota v každé buňce mapy a buňka s nejvyšší pravděpodobností představuje místo, kde se robot nejspíš nachází.

Jak jsme již v úvodu této kapitoly naznačili, algoritmus se skládá z hlavní smyčky, ve které se opakuje skenování a pohyb. Přesnější kroky algoritmu jsou následující:

1. Určení počátečních pravděpodobností
2. Hlavní smyčka
   I. Skenování
   II. Pohyb

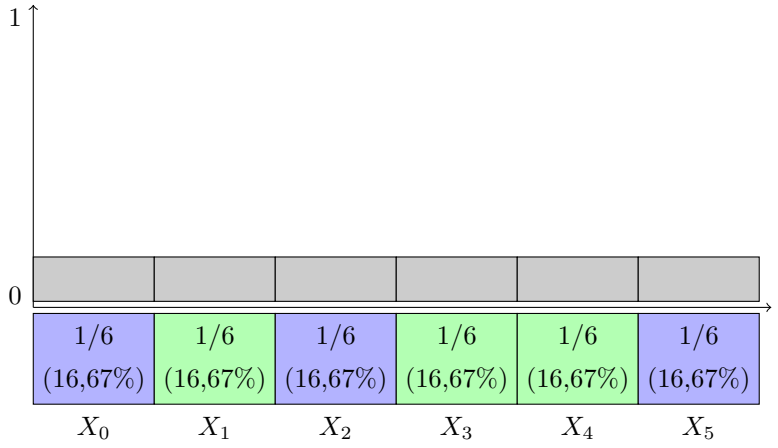
Pro zjednodušení si představme diskrétní mapu v 1D prostoru jako na obrázku 10 a předpokládejme, že robot dokáže rozeznat pomocí skeneru, zda se nachází na modrém nebo zeleném poli. Mapu považujme za cyklickou a tedy, pokud je robot na poloze X_5 a posune se o 1 políčko dopředu (doprava), tak se dostane na polohu X_0.



Obrázek 10: Mapa pro lokalizaci Monte Carlo

Určení počátečních pravděpodobností

Před prvním skenováním nemáme žádné tušení, na kterém políčku se robot nachází a tedy má každé políčko stejnou pravděpodobnost výskytu robota. Jedná se tedy o rovnoměrné rozdělení pravděpodobnosti. Tu vypočítáme pro i-té políčko jako $P(X_i) = 1/n$, kde n je celkový počet políček. V našem případě má tedy každé políčko pravděpodobnost $1/6$, jak je znázorněno histogramem na obrázku 11.



Obrázek 11: Mapa pro lokalizaci Monte Carlo po inicializaci

Skenování

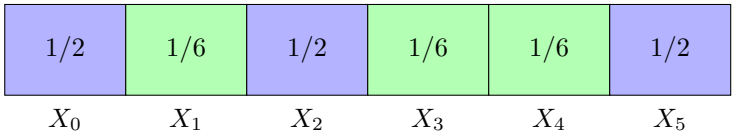
Nyní robot provede skenování a zjistí, že se nachází na modrém políčku. Protože nemáme bezchybný skener, tak nemůžeme jednoduše vyřadit ostatní (nemodrá) políčka. Ale můžeme jím snížit pravděpodobnost a zvýšit tak pravděpodobnost modrým políčkům. Pro příklad zaveďme pravděpodobnost, že robot naskenoval modrou barvu jako modrou $0,9$ a pravděpodobnost, že naskenoval zelenou barvu jako modrou $0,3$.

$$P(Z|X_i) = 0,9 \;\; pro\; i \in \{0,2,5\},$$

$$P(Z|X_i) = 0,3 \;\; pro\; i \in \{1,3,4\}$$

kde Z je prováděné měření, tedy zjištění, že je robot na modrém políčku.

Modrá políčka vynásobíme číslem $0,9$ a tedy $M_i = 0,9\,P(X_i)$ pro $i \in \{0,2,5\}$. U nemodrých políček vynásobíme pravděpodobnost číslem $0,3$, tedy $M_i = 0,3\,P(X_i)$ pro $i \in \{1,3,4\}$. Po provedení tohoto násobení nám vyjdou čísla na obrázku 12, kde v jednotlivých políčkách vidíme hodnoty M_i.



Obrázek 12: Mapa pro lokalizaci Monte Carlo po skenování

Podle věty o úplné pravděpodobnosti je součet všech pravděpodobností disjunktních jevů roven 1. Protože aktuální čísla v mapě nedávají součet 1, tak musíme provést normalizaci. A tedy pravděpodobnost $P(X_i)$ daného políčka po normalizaci je:

$$P(X_i) = \frac{M_i}{\sum_{j=0}^{n} M_j},$$

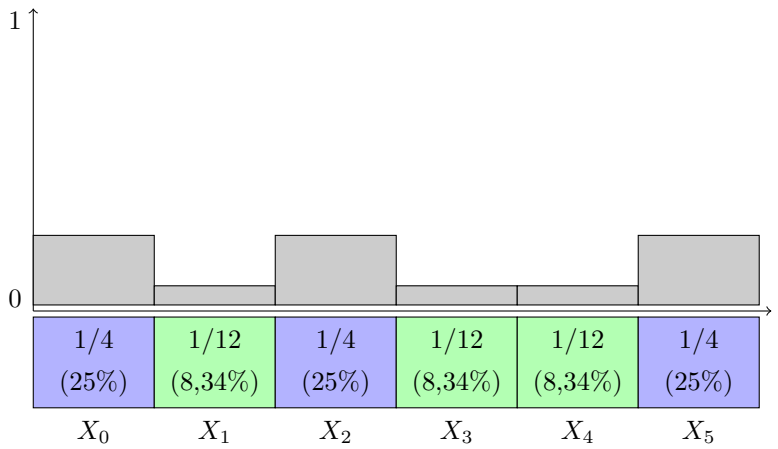
kde n je počet políček v mapě a M_i je hodnota i-tého políčka v mapě po vynásobení z předchozího kroku.

Těmito dvěma kroky jsme provedli výpočet pravděpodobnosti po skenování, tedy $P(X_i|Z)$ a využili jsme Bayesovu větu [20]:

$$P(X_i|Z) = \frac{P(Z|X_i)P(X_i)}{\sum_{j=0}^{n} P(Z|X_j)P(X_j)}$$

V prvním kroku jsme provedli násobení podle čitatele a ve druhém jsme výsledky podělili součinem ve jmenovateli.

Konkrétní rozložení pro tento příklad je zobrazeno na obrázku 13. Na něm vidíme, že původní rozdělení pravděpodobnosti bylo pro každé pole $1/6$ $(16,67\%)$, ale poté, co robot provedl skenování, se pravděpodobnost zvýšila u modrých políček na $1/4(25\%)$ a u zelených se snížila na $1/12$ $(8,34\%)$. V tuto chvíli ještě nevíme, kde se robot nachází, protože nejvyšší pravděpodobnost je na políčkách s indexy 0, 2 a 5.



Obrázek 13: Mapa pro lokalizaci Monte Carlo po normalizaci

Pohyb

Nyní robota posuneme o 1 políčko dopředu (doprava). Pokud bychom si byli absolutně jistí, že když zašleme robotovi příkaz *posuň se o 1 políčko dopředu*, tak se skutečně posune přesně o 1 políčko dopředu, tak by stačilo pravděpodobnosti v naší mapě posunout ve směru pohybu robota a rozložení pravděpodobnosti v mapě by tedy bylo určeno podle:

$$P(X_{(i+u) \bmod n}) = P(X_i),$$

kde u určuje, o kolik políček se má robot posunout (v našem případě $u = 1$). A protože jsme si určili mapu za cyklickou, tak musíme provádět operaci $modulo$.

Bohužel si nemůžeme být jistí, že se robot posune přesně o tolik políček, kolik mu určíme a tak musíme počítat i s možností, že se posunul o více, či méně políček. Určíme si tedy pravděpodobnost $0,8$, že se posunul skutečně o u políček a pravděpodobnost $0,1$, pokud se posunul o $u - 1$ nebo $u + 1$ políček a tedy:

$$P(X_{(i+u) \bmod n}|X_i) = 0,8$$

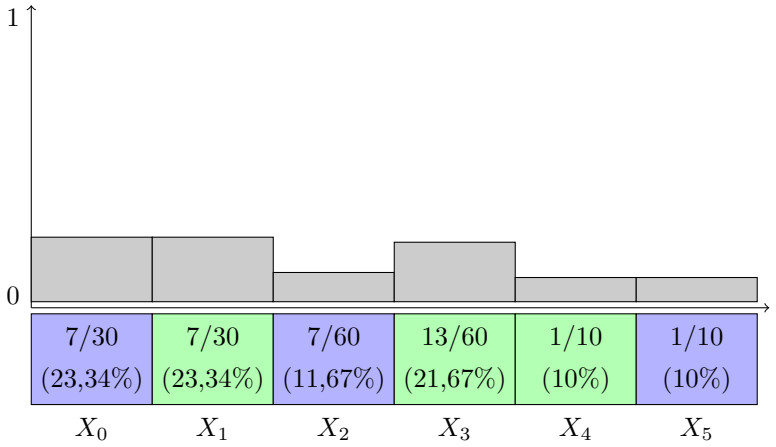
$$P(X_{(i+u-1) \bmod n}|X_i) = 0,1$$

$$P(X_{(i+u+1) \bmod n}|X_i) = 0,1$$

Pravděpodobnost i-tého políčka vypočítáme jako následující sumu:

$$P(X_i) = \sum_{j=-1}^{1} P(X_{(i+u+j) \bmod n}|X_i)\, P(X_{(i-u+j) \bmod n})$$

V našem příkladě se tedy pohyb na rozložení pravděpodobnosti projevil tak, jak je ukázáno na obrázku 14. Vidíme, že vysoké pravděpodobnosti jsou na polích X_0, X_1 a X_3. Což odpovídá tomu, že byl robot nejprve na modrém políčku a poté se posunul o 1 políčko doprava. Pravděpodobnost polohy robota ovšem klesla z původních 25% na $21,67\% - 23,34\%$, protože při pohybu došlo ke ztrátě informace o poloze robota.

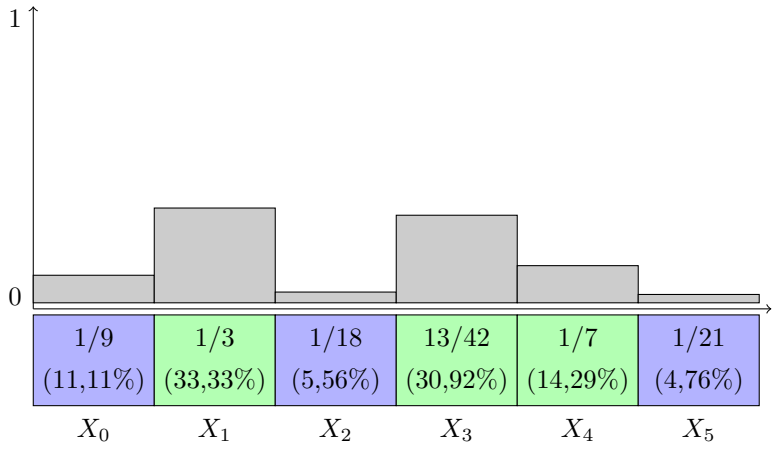


Obrázek 14: Mapa pro lokalizaci Monte Carlo po pohybu

Další iterace

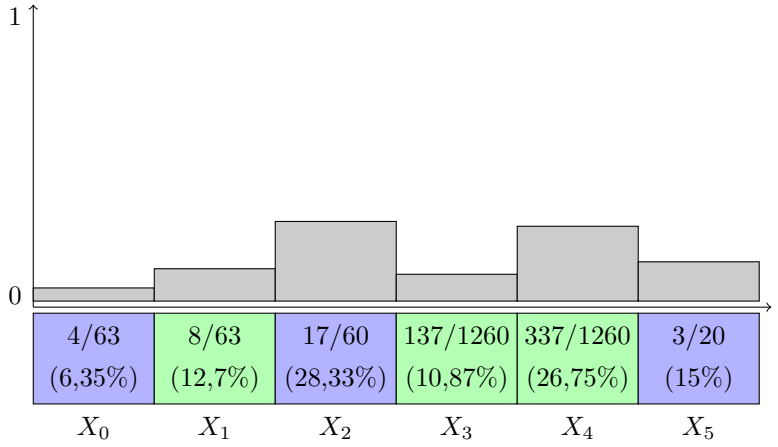
Aktuálně v hlavní smyčce algoritmu proběhla první iterace a nyní je na řadě opět skenování. Postup je shodný s tím, který je uveden v odstavci *Skenování* o pár řádků výše.

Nyní robot naskenuje zelenou barvu. Po přidání této informace do mapy a následné normalizaci vypadá mapa jako na obrázku 15. Vidíme, že se opět zvýšila pravděpodobnost zelených políček. Za zajímavost stojí všimnutí, že algoritmus nyní přiřazuje nižší pravděpodobnost políčku X_4, proti jiným zeleným políčkům a to díky tomu, že před tímto políčkem (vlevo) není políčko modré, což musí být na základě prvního skenování.



Obrázek 15: Mapa pro lokalizaci Monte Carlo po druhém skenování

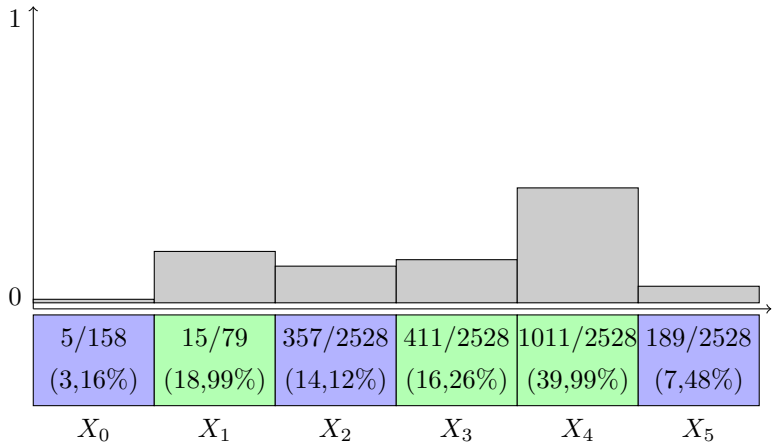
Dále následuje pohyb, který se opět provádí stejně jako v odstavci *Pohyb*. Posuneme robota o 1 políčko dopředu (doprava), tedy $u = 1$. Mapa po tomto pohybu je zobrazena na obrázku 16. Je zde zobrazeno, že opět došlo ke snížení pravděpodobností proti obrázku 15.



Obrázek 16: Mapa pro lokalizaci Monte Carlo po druhém pohybu

Nyní dojde zase ke skenování a robot naskenuje barvu zelenou, což ovlivní mapu tak, jak se ukázáno na obrázku 17. Kde lze vidět, že políčko X_4 má velmi vysokou pravděpodobnost $39,99\%$ a můžeme tedy usuzovat, že se robot nyní nachází právě na tomto políčku. Každý další pohyb by pravděpodobnost mírně snížil, ale následné skenování by pravděpodobnost opět zvýšilo.

Když si zrekapitulujeme akce, které robot vykonal, tak nejprve naskenoval modrou barvu, poté se posunul o jedno políčko dopředu, naskenoval zelenou barvu a následně se posunul znovu o 1 políčko dopředu a opět naskenoval barvu zelenou. Když se podíváme do mapy, tak vidíme, že robot začal v poloze X_2, kde naskenoval modrou barvu, poté se posunul na polohu X_3 a naskenoval zelenou a nakonec se posunul na polohu X_4, kde naskenoval opět zelenou a kde nám také vyšla největší pravděpodobnost, že se tam robot po dvou pohybech nachází.



Obrázek 17: Mapa pro lokalizaci Monte Carlo po třetím skenování

6.1.2 Particle filtr

Particle filtr [21], proti lokalizaci Monte Carlo, provádí lokalizaci na spojitém prostoru. Opět může mít více poloh stejnou pravděpodobnost (resp. váhu) výskytu robota. Zároveň tento algoritmus řeší i orientaci, kterou lokalizace Monte Carlo nepočítala.

Základní princip algoritmu je ve vytvoření n náhodných poloh a orientací robota v mapě, tzv. částic (particle) a pomocí měření ze skeneru se přiřadí váha každé částici. Čím více se přibližuje měření z robota a měření částice, tím větší váhu daná částice dostane. Následně se provede převzorkování, kdy se z původních n částic vyberou některé částice do nové kolekce o velikosti n, přičemž může být daná částice vybrána i několikrát. Pravděpodobnost, že bude daná částice vybrána závisí na její váze. Algoritmus se poté opakuje pro novou kolekci částic.

Vytvoření částic

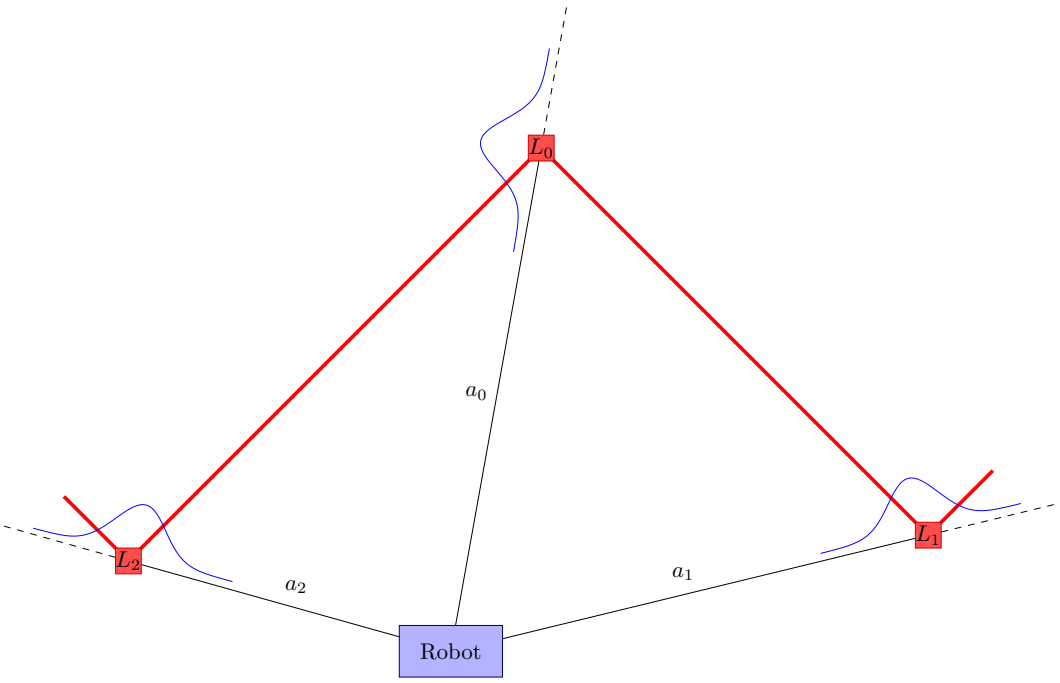
Prvním krokem Particle filtru je vytvoření n „robotů" (částic), které zaujímají náhodnou polohu a orientaci v mapě. Číslo n musí být dostatečně vysoké v závislosti na velikosti mapy a požadované výsledné přesnosti lokalizace. Čím více částic vytvoříme, tím je větší šance, že jedna z nich představuje našeho robota nebo se mu její poloha a orientace velmi přibližuje. Zvolme si tedy například $n = 1000$.

Přiřazení váhy částicím

Druhá část algoritmu určí, jak moc se jednotlivé částice podobají robotovi na základě měření. Mějme v mapě k význačných bodů, které dokáže skener identifikovat a jsou to body $L_0, L_1, \ldots, L_k$. Pro představu se může jednat o rohy v místnostech. Význačné body jsou podrobněji popsány v podkapitole 5.3. K těmto bodům dokážou jak robot, tak i částice změřit vzdálenost. Vzhledem k tomu, že skener neměří vzdálenosti přesně, tak musíme počítat s chybou měření. Ta je dána Gaussovou křivkou s hodnotami μ a σ^2, kde μ je umístěno přesně nad význačným bodem a σ^2 určuje velikost chyby měření. Tím zajistíme, že počítáme i s možností, že skener naměřil menší nebo větší vzdálenost.

Měření vzdáleností k význačným bodům včetně Gaussových křivek (modré křivky) je zobrazeno na obrázku 18. Červené čáry představují zdi a černé čáry $(a_0, a_1, \ldots, a_k)$ jsou

vzdálenosti od robota k bodům $L_0, L_1, \ldots, L_k$, které naměřil LIDARový skener, umístěný na robotovi. Výsledná vzdálenost význačného bodu od robota je ve skutečnosti o málo kratší nebo delší než a_i. Pravděpodobnost, která určuje tuto skutečnou vzdálenost, je určena právě zmíněnou Gaussovou křivkou.



Obrázek 18: Měření vzdáleností u Particle filtru z robota

Nyní s každou částicí provedeme měření vzdáleností k těmto význačným bodům. Na obrázku 19 je ukázáno, jak měření z robota sedí pro danou částici (zelené přerušované čáry) a kde by podle tohoto měření měly být význačné body $L'_0, L'_1, \ldots, L'_k$. Na první pohled je zřejmé, že tato částice není vhodným kandidátem a tak bude mít i nízkou váhu.

Měření z i-té částice je dáno hodnotami $b_{i,0}, b_{i,1}, \ldots, b_{i,k}$ (viz obr. 19). Samotná váha w_i se vypočítá podle Gaussovy křivky:

$$f(x) = ae^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

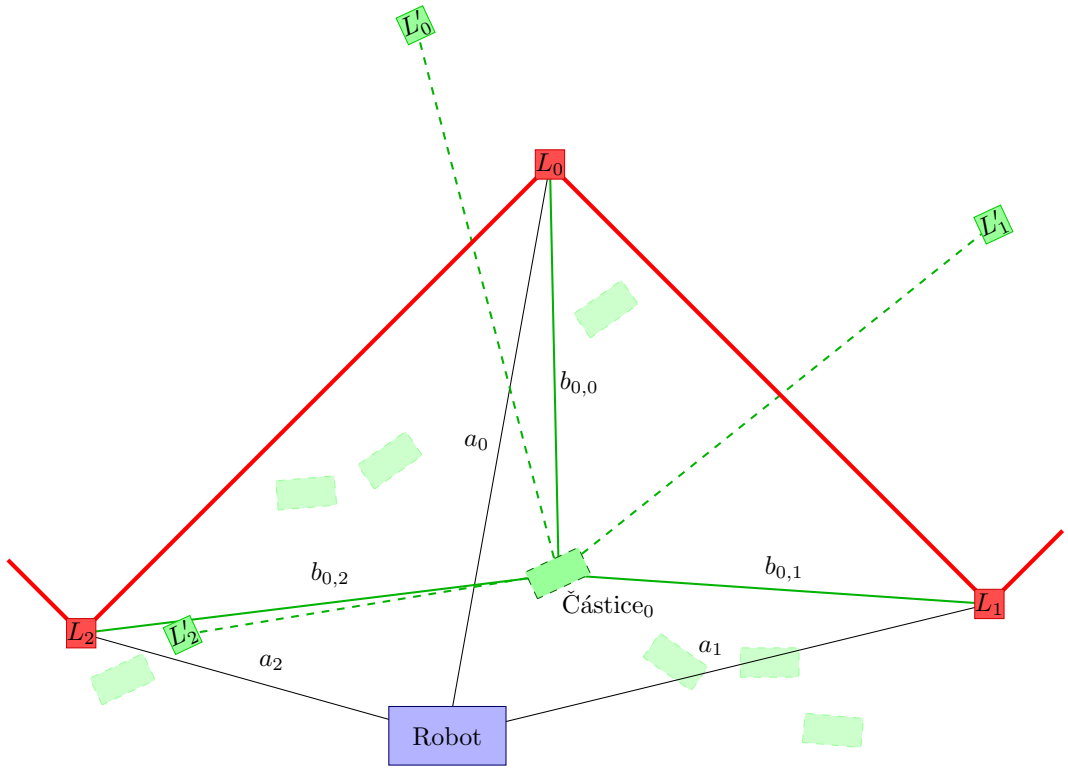
kde za a dosadíme $\frac{1}{\sqrt{2\pi\sigma^2}}$, za x hodnotu měření z i-té částice a za μ hodnotu měření z robota. Pro i-tou částici po dosazení vyjde vzorec:

$$w_i = \prod_{j=0}^{k} \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(b_{i,j}-a_j)^2}{2\sigma^2}} \right),$$

kde k je počet význačných bodů a σ^2 je chyba měření.

Převzorkování

Posledním krokem je převzorkování částic a to znamená, že do nové kolekce přiřadíme n částic ze staré kolekce s pravděpodobností α_i pro i-tou částici. V předchozím kroku jsme



Obrázek 19: Měření vzdáleností u Particle filtru z částice

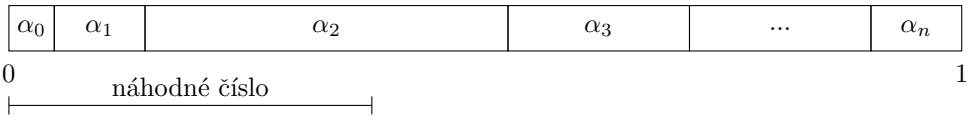
každé částici přiřadili váhu w_i. Abychom získali procentuální zastoupení jednotlivých částic, resp. určili pravděpodobnost, s jakou chceme jednotlivé částice vybírat, tak musíme váhy normalizovat. Normalizovaná váha, resp. výsledná pravděpodobnost výskytu i-té částice v nové kolekci, α_i se vypočítá podle:

$$W = \sum_{i=0}^{n} w_i,$$

$$\alpha_i = \frac{w_i}{W}.$$

Nyní náhodně vybereme n-krát částici ze staré kolekce a přiřadíme ji do nové kolekce. Můžeme si představit částice, resp. pravděpodobnosti jejich výběru, jako na obrázku 20, kde širší částici odpovídá větší pravděpodobnost. Poté si zvolíme náhodné číslo z rovnoměrného rozdělení pravděpodobnosti od 0 do 1. Abychom zjistili, do kterého intervalu náhodné číslo sahá, čili které částici odpovídá, tak budeme postupně inkrementovat $index$ o jedna, který začíná na 0, a z náhodného čísla budeme odečítat hodnotu α_{index}. Když se hodnota náhodného čísla dostane na 0 nebo bude menší než 0, tak do nové kolekce přiřadíme částici na pozici $index - 1$.

Vzhledem k tomu, že částice s větší pravděpodobností zabírá větší interval v celkovém intervalu $< 0; 1 >$ a částice s menší pravděpodobností, menší interval (ovšem v řádném poměru podle jejich pravděpodobností), tak dojde k výběru částic podle její pravděpodobnosti. Algoritmus převzorkování, resp. výběru částice podle její pravděpodobnosti, může vypadat jako ve výpisu kódu 2.



Obrázek 20: Pravděpodobnosti částic při převzorkování u Particle filtru

Výpis 2: Výběr částic podle jejich pravděpodobností

```
1   // Pole s pravdepodobnostmi vyberu castic
2   double[] alphas = { ... };
3
4   // Pole pro indexy novych castic
5   int [] newParticles = new int[alphas.length];
6
7   for(int i = 0; i < alphas.length; i++) {
8       double random = Math.random();
9       int index = 0;
10
11      while(random >= 0) {
12          random -= alphas[index];
13          index++;
14      }
15
16      newParticles[i] = index - 1;
17  }
```

Pohyb

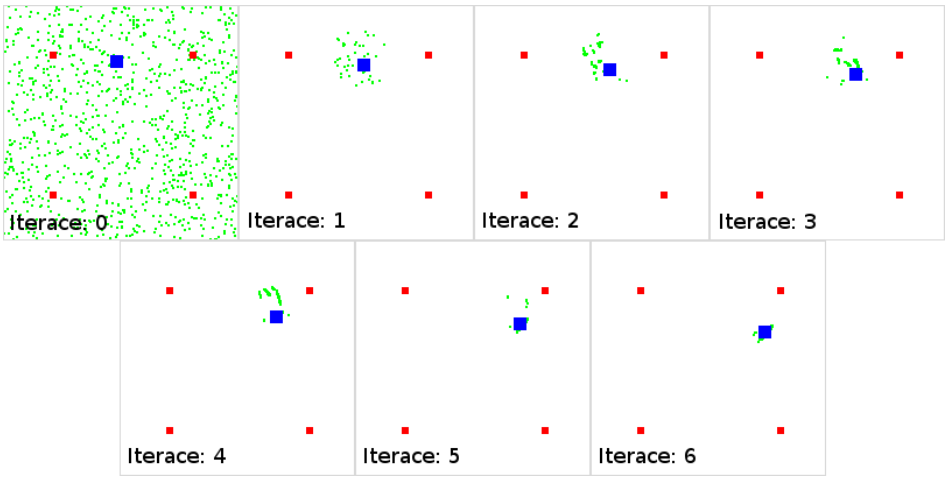
Jak jsme si na začátku kapitoly *Lokalizace* řekli, hlavní smyčka lokalizačních algoritmů představuje *skenování* a *pohyb*. Do této chvíle jsme popisovali Particle filtr v části skenování, které zvyšovalo pravděpodobnost míst, kde se nejpravděpodobněji robot nachází. Nyní je na řadě pohyb, který pravděpodobný výskyt robota mírně znepřesní.

Pohyb je u Particle filtru velice jednoduchý. Stačí, když každou částici posuneme pomocí stejného příkazu, podle kterého posuneme i robota. Nesmíme jen zapomenout k výsledným hodnotám Δx a Δy přidat chybu pohybu, tedy náhodné číslo, které se řídí normálním rozdělením pravděpodobnosti.

Další iterace

Po první iteraci Particle filtru se nám částice, které byli náhodně rozmístěné v mapě, začnou shlukovat do míst, kde je nejpravděpodobnější výskyt robota. Těchto shluků může být ze začátku více. S každou iterací však začnou shluky ubývat, až nakonec zůstane jen jeden a ten představuje výslednou polohu robota.

Na obrázku 21 je zobrazen průběh iterací u Particle filtru, kde červené čtverce představují význačné body. Vidíme, že v *iteraci 0* se nachází částice (zelené body) na celé mapě, ale už v *iteraci 1* se začnou částice více shlukovat kolem robota (modrý čtverec) a postupně se mu více přibližují. V *iteraci 5* a *6* jsou již téměř u robota a tak je zjištěna jeho lokalizace v mapě.



Obrázek 21: Průběh lokalizace pomocí Particle filtru

6.2 Mapování

Předpokládejme, že je náš robot vybaven velmi přesným GPS modulem, díky kterému dokáže zjišťovat svou polohu s přesností pár nanometrů. Nebo má velice přesný pohybový modul a tak pomocí odometrie (viz kapitola 4) můžeme v mapě zjistit posun Δx a Δy a orientaci $\Delta\alpha$ s chybou několika málo nanometrů. Poté bychom mohli pro každou polohu robota provést skenování LIDARovým systémem, údaje ze skeneru zpracovat a případně vizualizovat (více v podkapitole 5.1) a poskládat je do jedné mapy. Každý naskenovaný snímek by byl pouze posunut o Δx a Δy a případně natočen o úhel daný orientací podle $\Delta\alpha$ vůči předchozímu snímku a první snímek bychom brali jako výchozí.

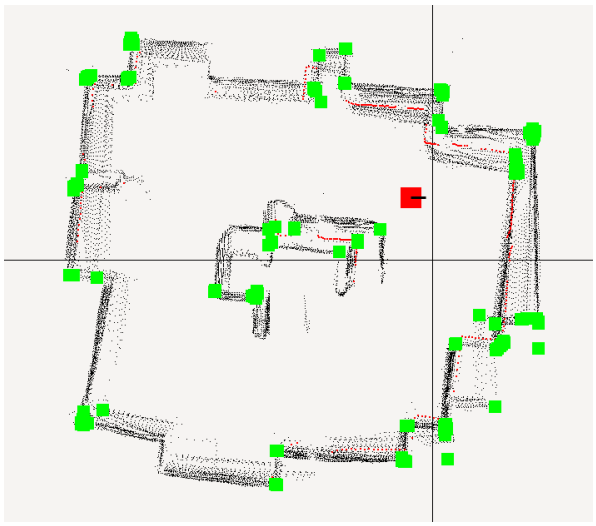
6.3 Problém současné lokalizace a mapování

V první podkapitole jsme si ukázali dvě techniky pro samostatnou lokalizaci. V obou dvou jsme měli k dispozici mapu, případně význačné body, pomocí kterých jsme lokalizaci mohli provést. Chyby způsobené měřením nebo pohybem robota jsme také vzali do úvahy a nakonec jsme byli schopni lokalizovat robota s určitou pravděpodobností a malou chybou. Základním předpokladem však byla mapa.

V některých případech se může robot dostat do neznámého místa, kde by se nemohl lokalizovat a tak potřebuje pro toto místo mapu vytvořit. Případně může být vyslán pouze za účelem mapování. V druhé podkapitole jsme popsali, jakým způsobem můžeme mapu vytvořit, ale hlavním a nejdůležitějším předpokladem byl přesný, až ideální lokalizační přístroj. Přístroje však mají chyby a tak nemůžeme získat dostatečně přesnou polohu robota, resp. LIDARového nosiče, abychom mohli jednotlivá naskenovaná data zapsat do jedné mapy.

Na obrázku 22 je ukázáno mapování robotem Neato XV-15, kdy se pro lokalizaci používala pouze odometrie. Vidíme, že po průjezdu celé mapy, kolem vnitřní překážky došlo k značnému znepřesnění polohy robota v mapě a tak jsou jednotlivé naskenované snímky posunuté a otočené. Čím déle se robot pohyboval, tím k větší kumulaci chyby došlo.

V případě, kdy neznáme mapu a potřebujeme robota v daném prostředí lokalizovat, at'už za účelem autonomního průjezdu prostředím nebo pro účely mapování, potřebujeme řešit



Obrázek 22: Mapování pomocí odometrie

problém současné lokalizace a mapování. Technika, která řeší tento problém se nazývá SLAM, což vychází z anglického názvu „Simultaneous localization and mapping" [22]. Konkrétních algoritmů, které používají tuto techniku je spousta, jedním z nich je i GraphSLAM, který si ukážeme v této práci.

7 Současná lokalizace a mapování - SLAM

Technika současné lokalizace a mapování, SLAM, řeší problém, kdy potřebujeme lokalizovat robota v neznámém prostředí, které souběžně mapujeme. Přesnější popis problému je popsán samostatně v podkapitole 6.3. Na obrázku 22 je zobrazena mapa předem neznámého prostředí, kterou robot vytvořil bez použití této techniky.

Hlavní problém je tedy způsoben nepřesností pohybového systému robota, kde zjišťujeme změnu polohy a orientace pomocí odometrie. Druhým problémem je chyba měření, která je sice znatelně menší než u pohybového systému, ale i tak se jí musíme zabývat. Technika SLAM dokáže řešit tyto problémy a jedná se tedy o způsob korekce chyb, které jsou v průběhu pohybu a skenování, bez použití této techniky, běžně kumulovány.

7.1 Korekce pomocí odometrie

Při korekci chyb pomocí SLAMu se spoléháme hlavně na měřicí přístroj, v našem případě na LIDARový skener, který je obecně daleko přesnější než pohybový systém. Mohou však nastat situace, kdy budou data ze skeneru nepoužitelná, protože se bude robot pohybovat v oblasti, kde nebude moct naskenovat význačné body, pomocí kterých by se mohl v prostředí lokalizovat. Dále také potřebujeme určit přibližnou změnu polohy a orientace robota, abychom měli prvotní odhad jeho nové polohy a orientace.

Pro tyto dva účely je nutné nejprve využít informace o změně polohy a orientace robota z odometrických dat, které můžeme vypočítat jako v kapitole 4. Díky těmto datům známe přibližnou polohu a orientaci robota a můžeme aktuální naskenovaný snímek posunout o vypočítané Δx, Δy a rotovat o $\Delta\alpha$, čímž se stejná místa v novém snímku a mapě téměř překryjí. Avšak nový snímek stále není na správné poloze, protože ani nová poloha robota není zatím správná.

7.2 Korekce pomocí spárování naskenovaných snímků - algoritmus ICP

Další způsob korekce polohy a orientace robota, a tedy i polohy a orientace nového naskenovaného snímku v mapě, je zjištění translační a rotační matice, která popisuje potřebnou rotaci a posun nového snímku, aby správně korespondoval se snímkem předchozím.

Algoritmus, který výpočet těchto matic zajišťuje se nazývá ICP [23] z anglického „Iterative Closest Point". Jedná se o iterativní algoritmus, kde dochází k postupné minimalizaci chyby, která popisuje rozdíl mezi novým a starým snímkem. Algoritmus se skládá z těchto kroků:

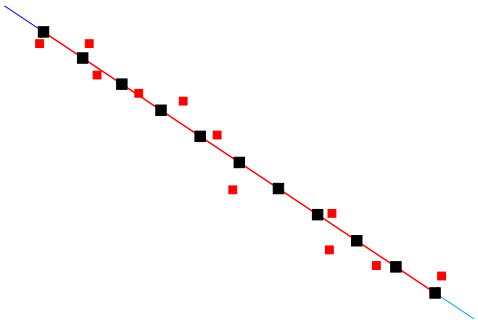
1. Výběr bodů.
2. Spojení bodů.
3. Ohodnocení každého spoje váhou.
4. Vyřazení některých spojů, podle jejich váhy.
5. Výpočet chyby podle váh zbylých spojů.
6. Minimalizace chyby pomocí translace a rotace nových bodů.

7.2.1 Výběr bodů

V prvním kroku vybereme některé body ze starého i nového snímku, pro které budeme korekci řešit. Starým snímkem myslíme ten, který byl naskenovaný přímo před novým snímkem. Můžeme také vybrat všechny body z obou snímků.

Protože nám LIDARový skener vrací body, které jsou umístěny okolo nějakých úseček při skenování rovné plochy (viz kapitola 5.2), tak je výhodnější zaměřit se na body, které leží přímo na těchto úsečkách. Pro algoritmus ICP tedy nebudeme brát přímo body, které nám poskytl LIDARový skener, ale zjistíme si body na přímkách, resp. úsečkách, které budou od sebe vzdáleny o určitou délku.

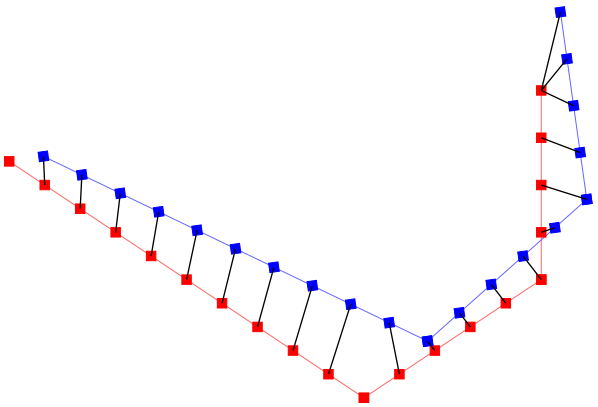
Na obrázku 23 vidíme původní červené body z LIDARového skeneru, které jsou umístěny kolem červené úsečky. Černou barvou jsou znázorněny body, které leží přímo na této úsečce. Tyto černé body si vypočítáme pro oba snímky a budeme je používat pro algoritmus ICP.



Obrázek 23: Převod LIDARových bodů na body na úsečce

7.2.2 Spojení bodů

V části pro spojení bodů dochází k nalezení nejbližšího bodu ze starého snímku ke každému bodu z nového snímku. Tím vytvoříme vazby, mezi novými a starými body, jak je tomu ukázáno na obrázku 24, kde červené body jsou staré body, modré body jsou nové a černé čáry znázorňují vytvořené vazby.



Obrázek 24: Spojení bodů pro algoritmus ICP

7.2.3 Ohodnocení každého spoje váhou

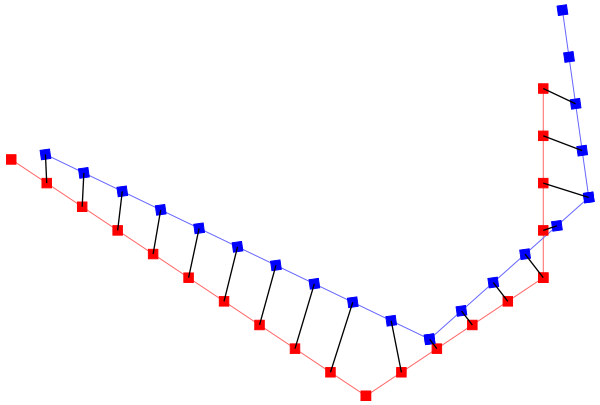
Protože potřebujeme určit, ke každému starému bodu právě jeden bod z nového snímku, tak musíme nějaké vazby vyřadit. To se děje v následujícím kroku, ovšem ještě předtím, musíme určit, které vazby zrušíme. Každé vazbě tedy přiřadíme váhu, jako čtverec vzdálenosti mezi body, které tvoří danou vazbu. Pro i-tou vazbu je pak dána váha w_i jako:

$$w_i = \| d_i - m_i \|^2 = (x_{d_i} - x_{m_i})^2 + (y_{d_i} - y_{m_i})^2,$$

kde d_i je starý bod i-té vazby, m_i je nový bod i-té vazby a kde x_{d_i} a y_{d_i} jsou souřadnice bodu d_i, a x_{m_i} a y_{m_i} jsou souřadnice druhého bodu v této vazbě, čili bodu m_i.

7.2.4 Vyřazení některých spojů, podle jejich váhy

Jak je z obrázku 24 patrné, může být jeden bod ze starého snímku (červená barva) ve vazbě s více novými body. Situace je vidět v pravém horním rohu obrázku. V tomto kroku každé skupině těchto vazeb ponecháme vždy jen jednu a to takovou, která má nejmenší váhu, čili nejmenší vzdálenost mezi body ve vazbě. Výsledek tohoto kroku je patrný na obrázku 25, kde v našem případě došlo pouze k odstranění dvou vazeb ze zmíněného pravého horního rohu.



Obrázek 25: Vyřazení spojů u algoritmu ICP

7.2.5 Výpočet chyby podle váh zbylých spojů

Samotnou chybu err mezi starým a novým snímkem můžeme vypočítat jako součet váh jednotlivých vazeb a tedy:

$$err = \sum_{i=0}^{n} w_i = \sum_{i=0}^{n} \| d_i - m_i \|^2,$$

přičemž n určuje celkový počet vazeb a w_i je váha i-té vazby, vypočítaná podle třetího kroku algoritmu. Z intuitivního pohledu vidíme, že když budou body ve vazbě blíže k sobě, tak bude i jejich váha menší a tedy i celková chyba bude menší. Ideální případ tvoří chyba blízká nule, kde by byly body z obou snímků na téměř stejných polohách.

7.2.6 Minimalizace chyby pomocí translace a rotace nových bodů

V posledním kroku algoritmu jde o nalezení takové rotační matice R a translační matice T, které po aplikování na body m_i sníží chybu err na minimum [24]. Výpočet chyby tedy můžeme přepsat jako:

$$err = \sum_{i=0}^{n} \parallel d_i - (Rm_i + T) \parallel^2 .$$

Jednou z metody pro nalezení takových matic R a T je metoda singulárního rozkladu matice (SVD).

Výpočet rotační matice

Pokud si představíme dva polygony, na kterých leží body d_i a m_i, pak pro minimální chybu musí platit, že jejich centroidy jsou na totožné poloze. Vzdálenosti mezi jednotlivými body d_i a centroidem $\bar{d}$ jsou pak dány podle d_{c_i}. Pro body m_i platí to stejné, čili:

$$\bar{d} = \frac{1}{n} \sum_{i=0}^{n} d_i,$$

$$d_{c_i} = d_i - \bar{d},$$

$$\bar{m} = \frac{1}{n} \sum_{i=0}^{n} m_i,$$

$$m_{c_i} = m_i - \bar{m}.$$

Výpočet chyby pak můžeme pro rotaci přepsat jako:

$$err = \sum_{i=0}^{n} \parallel d_{c_i} - Rm_{c_i} \parallel^2 = \sum_{i=0}^{n} \left(d_{c_i}^T d_{c_i} + m_{c_i}^T m_{c_i} - 2 d_{c_i}^T R m_{c_i} \right).$$

K minimalizaci této rovnice dojde, pokud bude poslední člen rovnice maximální, což je ekvivalentní k maximalizaci stopy $Tr(RH)$, kde H je korelační matice definována jako:

$$H = \sum_{i=0}^{n} m_{c_i} d_{c_i}^T.$$

Pokud je singulární rozklad matice H dán jako $H = U\Sigma V^T$, pak rotační matice R se vypočítá ze vztahu:

$$R = VU^T.$$

Výpočet translační matice

Ideální translační matice pro posunutí centroidu $\overline{m}$ na polohu centroidu $\overline{d}$ je dána podle výše uvedené rovnice pro výpočet chyby jako:

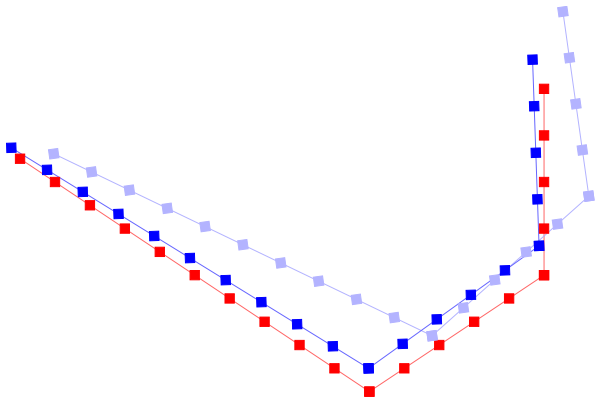
$$T = \bar{d} - R\bar{m}.$$

Aplikace rotační a translační matice

Po výpočtu těchto dvou matic provedeme rotaci a translaci každého bodu m_i^t z nového snímku podle:

$$m_i^{t+1} = Rm_i^t + T.$$

Tím získáme novou sadu bodů m_i^{t+1}, které se blíže přibližují bodům ze starého snímku, jak je zobrazeno na obrázku 26, kde původní body nového snímku m_i^t jsou znázorněny světle modrou barvou a nové body m_i^{t+1} jsou nakresleny tmavě modrou barvou.



Obrázek 26: Použití rotace a translace z algoritmu ICP pro body nového snímku

7.2.7 Další iterace algoritmu ICP

Po aplikaci rotační a translační matice se body z nového snímku m_i více přiblížili bodům ze starého snímku d_i a došlo tak k minimalizaci chyby err. Algoritmus se nyní opakuje od kroku 2 (spojení bodů), pokud při jednotlivých iteracích dochází ke zmenšení chyby err. Pokud je chyba v následující iteraci větší, než chyba v iteraci předešlé, tak algoritmus končí.

Výsledkem je pak několik rotačních a translačních matic R_i a T_i. Ty po postupném aplikování na polohu a orientaci robota, dokážou zpřesnit Δx, Δy a $\Delta\alpha$ posledního pohybu robota. Po této korekci polohy a orientace můžeme posunout nový snímek o přesnější Δx a Δy a rotovat jej o $\Delta\alpha$, čímž se bude nový snímek více korespondovat se starým snímkem, než tomu bylo při samostatném použití odometrických dat.

7.3 Spárování význačných bodů

V některých algoritmech řešících lokalizaci, jako je například Particle filtr (viz kapitola 6.1.2) nebo v určitých algoritmech typu SLAM (např. GraphSLAM, jejichž jednou částí je i lokalizace), se určuje poloha LIDARového nosiče díky význačným bodům. V podkapitole 5.3 jsme si ukázali, jak můžeme tyto body pomocí LIDARových dat najít a zároveň jsme řekli, že jsou tyto body specifické díky své vlastnosti, která určuje, že můžeme najít význačný bod L_i ze starého snímku (případně z mapy) v novém snímku, pořízeném LIDARovými skenerem.

7.3.1 Metoda nejbližšího souseda

Jednou z technik, jak tyto význačné body propojit v rámci několika snímků, je metoda „nejbližšího souseda". Tato metoda předpokládá, že jsou korespondující význačné body blízko sebe, což je splněno po provedení korekce pomocí ICP, kterou předcházela korekce pomocí odometrie. Dále stačí postupně projít význačné body v novém snímku a najít v mapě takový význačný bod, jehož vzdálenost je k danému bodu z nového snímku nejmenší.

7.3.2 Metoda korespondujících vzdáleností

Další technika, která se může použít pro nalezení korespondujících význačných bodů, využívá vzdálenosti, které jsou mezi význačnými body. Tyto vzdálenosti jsou mezi význačnými body vždy stejné, resp. velmi podobné, at'už jsou to body z aktuálního nebo předešlého snímku. Jedinými podmínkami je, aby nebyly význačné body symetricky rozmístěny a aby v každém z těchto dvou snímků byly alespoň 3 stejné význačné body.

Prvním krokem algoritmu je vytvoření vzdálenostních tabulek pro starý a nový snímek. Mějme význačné body $L_0, L_1, L_2, \ldots, L_n$, kde $n > 2$ ze starého snímku a body $L'_0, L'_1, L'_2, \ldots, L'_m$, kde $m > 2$ z nového snímku. Vzdálenostní tabulka pro starý snímek pak vypadá následovně. Stejným způsobem vytvoříme i vzdálenostní tabulku pro nový snímek.

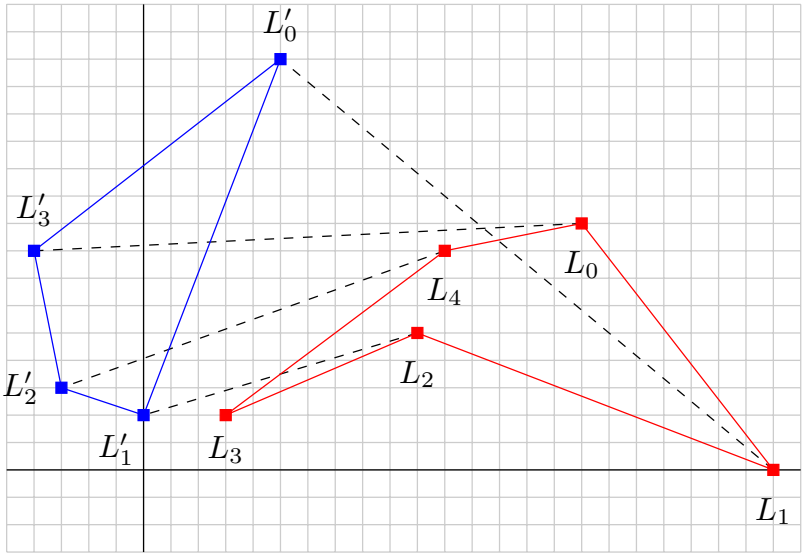
	1	2	3	$\cdots$	k
L_0	$\parallel L_0 - L_1 \parallel$	$\parallel L_0 - L_2 \parallel$	$\parallel L_0 - L_3 \parallel$	$\cdots$	$\parallel L_0 - L_{k+0} \parallel$
L_1	$\parallel L_1 - L_2 \parallel$	$\parallel L_1 - L_3 \parallel$	$\parallel L_1 - L_4 \parallel$	$\cdots$	$\parallel L_1 - L_{k+1} \parallel$
L_2	$\parallel L_2 - L_3 \parallel$	$\parallel L_2 - L_4 \parallel$	$\parallel L_2 - L_5 \parallel$	$\cdots$	$\parallel L_2 - L_{k+2} \parallel$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_n	$\parallel L_n - L_0 \parallel$	$\parallel L_n - L_1 \parallel$	$\parallel L_n - L_2 \parallel$	$\cdots$	$\parallel L_n - L_{k-1} \parallel$

Tabulka 1: Vzdálenostní tabulka mezi význačnými body

Ve vzdálenostní tabulce každému řádku odpovídá jeden význačný bod a ve sloupcích se nachází vzdálenosti mezi daným význačným bodem a mezi následujícími k sousedy, přičemž se sousedi počítají cyklicky. Což znamená, že pro předposlední bod L_{n-1} je třetím sousedem bod L_1.

V konkrétním případě mějme význačné body starého snímku $L_0 = [16; 9]$, $L_1 = [23; 0]$, $L_2 = [10; 5]$, $L_3 = [3; 2]$ a $L_4 = [11, 8]$, jak jsou zobrazeny na obrázku 27 červenou barvou a déle body nového snímku $L'_0 = [5, 1; 14, 9]$, $L'_1 = [-0, 1; 1, 9]$, $L'_2 = [-3, 1; 3, 1]$ a $L'_3 = [-3, 9; 8, 1]$, které jsou znázorněny modrou barvou. Vzdálenostní tabulky pro tyto význačné body vypadají následovně (tabulka 2 a 3). Hodnoty vzdáleností jsou zaokrouhleny na jedno desetinné místo. Vidíme, že i přes to, že by měla vzdálenostní tabulka pro nové body (tabulka 3) obsahovat pouze hodnoty z tabulky pro staré body (tabulka 2), tak tomu tak není. Rozdílné vzdálenosti jsou způsobeny chybou měření skeneru.

Druhým krokem algoritmu je nalezení páru korespondujících bodů pomocí vzdálenostních tabulek. Když se podíváme na bod L_0 a jeho korespondující bod L'_3 z obrázku 27, tak vidíme, že vzdálenosti k sousedním bodům jsou téměř stejné. Chceme tedy najít nejdelší společnou podposloupnost (LCS) těchto vzdáleností, kde za totožné vzdálenosti bereme ta-



Obrázek 27: Spárování význačných bodů pomocí korespondujících vzdáleností

	1	2	3	4
L_0	$11,4$	$7,2$	$14,8$	$5,1$
L_1	$13,9$	$20,1$	$14,4$	$11,4$
L_2	$7,6$	$3,2$	$7,2$	$13,9$
L_3	$10,0$	$14,8$	$20,1$	$7,6$
L_4	$5,1$	$14,4$	$3,2$	$10,0$

Tabulka 2: Vzdálenostní tabulka pro staré body

	1	2	3
L'_0	$14,0$	$14,4$	$11,3$
L'_1	$3,2$	$7,3$	$14,0$
L'_2	$5,1$	$14,4$	$3,2$
L'_3	$11,3$	$7,3$	$5,1$

Tabulka 3: Vzdálenostní tabulka pro nové body

kové, které jsou velmi podobné. Tedy $|l_0 - l_1| < tolerance$, kde $tolerance$ je například $0,2$ a hodnoty l_0 a l_1 jsou vzdálenosti mezi význačnými body.

Pro bod L_0 je posloupnost vzdáleností k sousedním bodům z tabulky 2: $11,4$; $7,2$; $14,8$; $5,1$ a s posloupnostmi L'_0, L'_1 a L'_2 má nejdelší společnou podposloupnost délky 0. Ovšem s posloupností bodu L'_3, tedy $11,3$; $7,3$; $5,1$, je nejdelší společná podposloupnost 2. Protože je mezi těmito body nejdelší společná posloupnost největší délky, tak tyto body označíme za korespondující.

Stejná situace nastává také pro body L_1 a L'_0, L_2 a L'_1, L_4 a L'_2. Zajímavé je hledání nejdelší společné podposloupnosti pro bod L_3, který v obrázku 27 nemá korespondující bod.

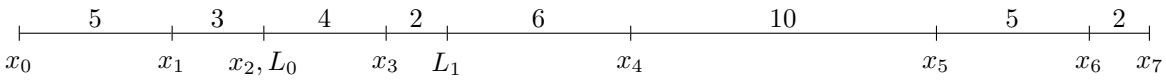
V takovém případě vyjde LCS délky 0 pro všechny body, takže můžeme prohlásit, že tento bod nemá korespondující bod.

7.4 GraphSLAM

Algoritmus GraphSLAM [25] je jedním z algoritmů, které řeší problém současné lokalizace a mapovaní pomocí techniky SLAM. Hlavním cílem algoritmu je zpřesnit polohy robota, které v průběhu své cesty navštívil a také určit přesnější polohu význačných bodů, které z určitých poloh naskenoval.

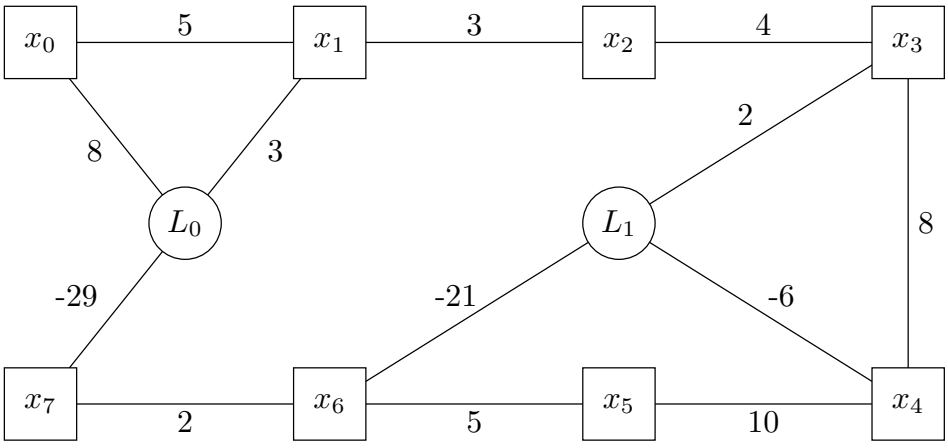
7.4.1 Vytvoření grafu

Pro zjednodušení si představme 1D prostor, ve kterém se robot nachází, pohybuje a měří vzdálenosti k význačným bodům. Řešení GraphSLAMu s více dimenzemi bude později popsáno na konci této kapitoly, ovšem základní princip je stejný i pro jednu dimenzi. Tento 1D prostor, resp. mapa může vypadat jako na obrázku 28, kde vidíme jednotlivé polohy robota $x_0, x_1, \ldots, x_7$ a význačné body L_0 a L_1 spolu s vzdálenostmi mezi nimi. Pokud tedy robot z polohy x_0 naskenuje význačný bod L_0, tak zjistí, že se od něj nachází ve vzdálenosti 8.



Obrázek 28: Mapa pro GraphSLAM v 1D

Tuto mapu si můžeme také představit jako ohodnocený graf, jehož uzly jsou dány polohami robota $x_0, x_1, \ldots, x_7$ a význačnými body L_0 a L_1 a jehož hrany popisují vzdálenosti mezi těmito polohami a význačnými body. Graf je zobrazen na obrázku 29 a vidíme, že například význačný bod L_1 byl naskenován pouze z poloh x_3, x_4 a x_6.

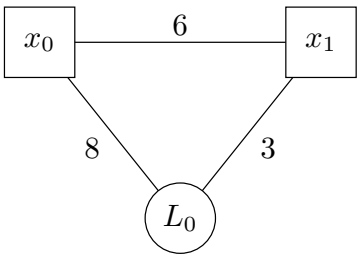


Obrázek 29: Graf mapy pro GraphSLAM

Takto by ovšem vypadal graf, pokud bychom měli ideální pohybový systém a ideální skener u robota. Robot začíná na poloze x_0, kde naskenuje k význačnému bodu L_0 vzdálenost 8, poté robotovi pošleme příkaz, aby se posunul o 5 jednotek a po dokončení tohoto pohybu

robot změří ke stejnému význačnému bodu L_0 vzdálenost 3. V tomto případě bychom SLAM vůbec nemuseli řešit a robot by se dokázal lokalizovat a zároveň by i vytvořil přesnou mapu prostředí.

U reálného robota však situace vypadá jinak. Pohybový systém i skener mají chyby a tak pohyb z polohy x_0 do x_1 a skenování z těchto poloh bude vypadat spíše jako na obrázku 30. Robot zde z polohy x_0 naskenuje vzdálenost 8 k význačnému bodu L_0, stejně jako v předchozím případě. Nyní mu přikážeme, aby se posunul o 6 jednotek a po tomto pohybu nám vrátí k bodu L_0 vzdálenost 3.



Obrázek 30: Část grafu pro GraphSLAM znázorňující realitu

Z intuitivního pohledu je zřejmé, že je něco špatně. Pokud se robot posunul o 6 jednotek, pak by při druhém skenování měl naměřit vzdálenost 2 a nikoli 3, nebo by první naměřená vzdálenost měla být 9, což by pak po uskutečnění pohybu odpovídalo. Tím jsme ukázali, že možná skener způsobuje chybu měření a tak poloha bodu L_0 není přesná. Další možností může být chyba pohybového systému, který robota neposunul o 6 jednotek, jak jsme mu přikázali, ale pouze o 5.

7.4.2 Vytvoření rovnic

Pokud se zaměříme na část grafu z obrázku 30, tak můžeme jednotlivé polohy a význačný bod zapsat pomocí následujících rovnic. Společně s určením počáteční polohy, pak získáme soustavu rovnic:

$$\begin{aligned} x_0 &= 0 \\ x_0 + 6 &= x_1 \\ x_0 + 8 &= L_0 \\ x_1 + 3 &= L_0 \end{aligned}$$

V reálném případě však např. poloha x_1 není pouze bod posunutý o 6 od polohy x_0, nýbrž se jedná o normální rozdělení pravděpodobnosti kolem bodu x_1 s určitým rozptylem σ_p^2, který popisuje chybu pohybového systému. Konkrétně tedy:

$$f(x) = ae^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma_p^2}} = ae^{-\frac{1}{2}\frac{(x_0+6-x_1)^2}{\sigma_p^2}}.$$

Stejně je tomu i pro měření s chybou σ_m^2. Význačný bod je na pozici okolo bodu L_0 podle prvního měření:

$$ae^{-\frac{1}{2}\frac{(x_0+8-L_0)^2}{\sigma_m^2}}$$

a podle druhého měření:

$$ae^{-\frac{1}{2}\frac{(x_1+3-L_0)^2}{\sigma_m^2}}.$$

Výsledné rozdělení pravděpodobnosti je pak dáno součinem dílčích pravděpodobností a tedy:

$$ae^{-\frac{1}{2}\frac{(x_0+8-L_0)^2}{\sigma_m^2}}\ ae^{-\frac{1}{2}\frac{(x_1+3-L_0)^2}{\sigma_m^2}}.$$

Pro výpočet maxima funkce, které je ve střední hodnotě μ normálního rozdělení, v našem případě L_0, vůbec nepotřebujeme konstanty a, protože ty neovlivňují střední hodnotu μ. Dále můžeme odstranit exponenciální členy, pokud součin změníme na součet a v poslední řadě i $-\frac{1}{2}$. Tím nám, po odmocnění, zbude:

$$\frac{x_0}{\sigma_m}+\frac{8}{\sigma_m}-\frac{L_0}{\sigma_m}+\frac{x_1}{\sigma_m}+\frac{3}{\sigma_m}-\frac{L_0}{\sigma_m}.$$

Maximum, což je μ, dosahuje Gaussova křivka, když $\mu - x = 0$. V našem případě tedy můžeme dát předchozí vztah rovný nule, abychom vypočítali L_0. Po úpravě nám vyjde rovnice:

$$-\frac{x_0}{\sigma_m}-\frac{x_1}{\sigma_m}+2\frac{L_0}{\sigma_m}=\frac{8}{\sigma_m}+\frac{3}{\sigma_m}.$$

Stejné operace můžeme provést i pro výpočet x_0 a x_1, a společně s výše uvedenou rovnicí nám vyjde soustava rovnic:

$$\begin{aligned}
\left(\frac{1}{\sigma_m}+\frac{1}{\sigma_m}+\frac{1}{\sigma_p}\right)x_0+\left(-\frac{1}{\sigma_p}\right)x_1+\left(-\frac{1}{\sigma_m}\right)L_0 &= -\frac{1}{\sigma_m}8+\left(-\frac{1}{\sigma_p}\right)6\\
-\frac{1}{\sigma_p}x_0+\left(\frac{1}{\sigma_p}+\frac{1}{\sigma_m}\right)x_1+\left(-\frac{1}{\sigma_m}\right)L_0 &= \frac{1}{\sigma_p}6+\left(-\frac{1}{\sigma_m}\right)3\\
-\frac{1}{\sigma_m}x_0+\left(-\frac{1}{\sigma_m}\right)x_1+\left(\frac{1}{\sigma_m}+\frac{1}{\sigma_m}\right)L_0 &= \frac{1}{\sigma_m}8+\frac{1}{\sigma_m}3.
\end{aligned}$$

Tuto soustavu můžeme přepsat do tvaru $\Omega\mu = \xi$, kde v matici Ω budou chyby měření σ_m a chyby pohybu σ_p, vektor μ bude obsahovat neznámé polohy robota a polohy význačných bodů a vektor ξ bude obsahovat hodnoty na pravé straně rovnice. Konkrétně tedy:

$$\begin{bmatrix}
\left(\frac{1}{\sigma_m}+\frac{1}{\sigma_m}+\frac{1}{\sigma_p}\right) & \left(-\frac{1}{\sigma_p}\right) & \left(-\frac{1}{\sigma_m}\right)\\
\left(-\frac{1}{\sigma_p}\right) & \left(\frac{1}{\sigma_p}+\frac{1}{\sigma_m}\right) & \left(-\frac{1}{\sigma_m}\right)\\
\left(-\frac{1}{\sigma_m}\right) & \left(-\frac{1}{\sigma_m}\right) & \left(\frac{1}{\sigma_m}+\frac{1}{\sigma_m}\right)
\end{bmatrix}
\begin{bmatrix} x_0\\ x_1\\ L_0 \end{bmatrix}=
\begin{bmatrix}
-\frac{1}{\sigma_m}8+\left(-\frac{1}{\sigma_p}\right)6\\
\frac{1}{\sigma_p}6+\left(-\frac{1}{\sigma_m}\right)3\\
\frac{1}{\sigma_m}8+\frac{1}{\sigma_m}3
\end{bmatrix}$$

Pro výpočet vektoru neznámých hodnot μ, pak platí $\mu = \Omega^{-1}\xi$. Když si určíme chybu měření $\sigma_m = 2$ a chybu pohybu daleko větší $\sigma_p = 18$, tak nám po vyřešení soustavy rovnic vyjde:

$$x_0 = 0$$

$$x_1 = 5,18$$

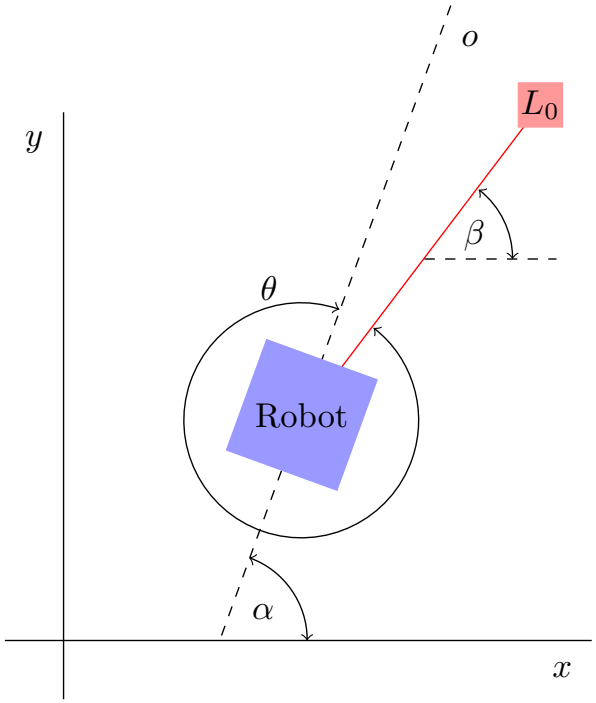
$$L_0 = 8,09.$$

Tím jsme určili střední hodnoty x_0, x_1 a L_0, což jsou místa s největší pravděpodobností jejich výskytu. Proti hodnotám z části grafu na obrázku 30 vidíme, že algoritmus správně zjistil, že poloha x_1 není 6, ale je někde kolem bodu $5,18$, protože na výsledné pravděpodobnosti se více podílí údaje ze skeneru, protože mají nastavenou menší chybu.

7.5 Výpočet orientace u GraphSLAMu

Výše uvedeným způsobem dokážeme pomocí GraphSLAMu vypočítat polohy význačných bodů a také polohy robota. K úplné znalosti systému však potřebujeme znát i orientaci robota ve všech polohách, které navštívil. Tuto orientaci využijeme pro nejpravděpodobnější rotaci naskenovaných snímků, aby vytvořili celistvou mapu.

Díky tomu, že nám GraphSLAM zpřesní polohy význačných bodů a robota, a také víme, pod jakým úhlem byly naskenovány význačné body v každé poloze robota, tak můžeme orientaci zpětně dopočítat. U každého naskenovaného (případně vypočítaného) význačného bodu víme, pod jakým úhlem byl vyslán paprsek, který tento bod zaznamenal. Na obrázku 31 jde o úhel θ pro naskenovaný význačný bod L_0. Po zpřesnění polohy robota (modrý čtverec) a polohy význačného bodu L_0 také můžeme vypočítat úhel β. Výsledná orientace α, kterou musí robot mít, aby mohl ze své pozice naskenovat bod L_0 pod úhlem θ, pak je dána podle $\alpha = \beta - \theta$.



Obrázek 31: Výpočet úhlu pro GraphSLAM

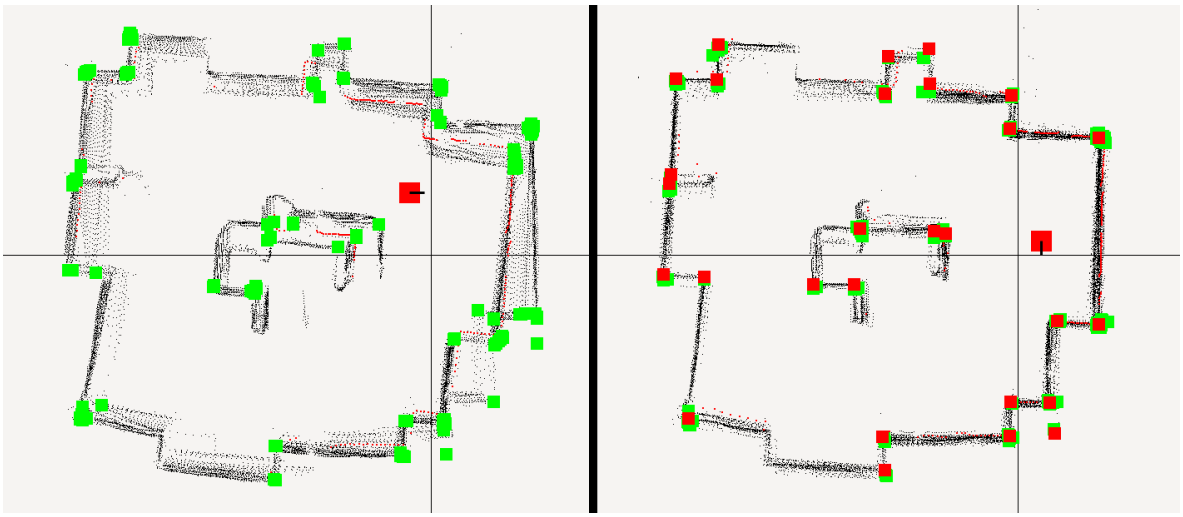
Tím jsme určili orientaci podle jednoho význačného bodu. V každé poloze však mohl robot naskenovat více význačných bodů, takže nám vyjde více úhlů α_i. Výsledný úhel $\bar{\alpha}$, pro tuto polohu robota, pak můžeme vypočítat pomocí průměrů těchto úhlů [26]. Stačí, když si úhly z polární soustavy souřadnic převedeme do kartézské soustavy souřadnic jako body na jednotkové kružnici. U těchto bodů spočítáme aritmetický průměr a výsledný bod převedeme zpět od polární soustavy souřadnic. Konkrétně pak podle:

$$\bar{\alpha} = \operatorname{atan2}\left(\frac{1}{n}\sum_{i=0}^{n}\sin\alpha_i, \frac{1}{n}\sum_{i=0}^{n}\cos\alpha_i\right).$$

7.6 Reálné použití GraphSLAMu

LIDARový skener u robota Neato XV-15 snímá prostor v rovině a jedná se tedy o 2D skener. V kapitole 7.4 jsme řešili GraphSLAM pro jednu dimenzi. Řešení pro druhou a další dimenzi spočívá pouze ve vytvoření více matic Ω a vektorů ξ pro každou dimenzi.

Na následujícím obrázku 32 vidíme v levé části zmapovaný prostor robotem Neato XV-15 pouze pomocí odometrie a v pravé části vidíme tentýž prostor zmapovaný pomocí algoritmu GraphSLAM. Zelené body představují význačné body (rohy) ze skenování a červenou barvu mají význačné body po korekci GraphSLAMem. Rozdíl v přesnosti výsledné mapy je velmi znatelný, zvlášť když se zaměříme na pravou stěnu, kde mapování pomocí odometrie vytvořilo dvě stěny, které byly od sebe velmi vzdálené, ale při použití GraphSLAMu došlo k přesnějšímu mapování a zeď se jeví správně jako jedna čára.



Obrázek 32: Rozdílné mapování pomocí odometrie a pomocí algoritmu GraphSLAM

8 Autonomní mapování

Do této chvíle jsme řešili tzv. pasivní SLAM, kdy jsme pohyb robota ovládali ručně a měnili jsme jeho polohu a orientaci např. pomocí šipek na klávesnici. Častou úlohou, při které se využívá současná lokalizace a mapování, je autonomní řízení vozidel, kdy si robot sám určuje cíl své cesty. V tomto případě již hovoříme o aktivním SLAMu.

Autonomní mapování má za cíl projet všemi místy v předem určené oblasti a pro tuto oblast vytvořit mapu. Zároveň se robot musí umět vyhýbat překážkám, které v průběhu mapování objeví a rozhodnout o cílech a cestách, po kterých bude jezdit. Tyto cesty a cíle by měl také volit efektivně, aby ujel krátkou vzdálenost a zmapoval tak největší prostor.

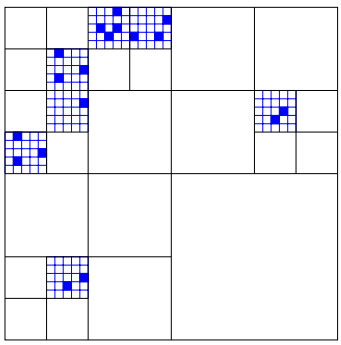
V této kapitole se budeme zabývat pouze určováním cílů, které musí robot navštívit, aby zmapoval celou určenou oblast. Samotné řešení nejkratší a bezpečné cesty, která dostane robota z počáteční polohy bodu A do cílového bodu B, je náplní následující kapitoly 9.

8.1 Mapa pomocí mřížky s kvadrantovým stromem

Před samotným mapováním si musíme určit strukturu, která nám bude reprezentovat mapu. Jednou z možností je mřížka s určitým rozlišením, čili s určitou velikostí čtvercových buněk. Do těchto buněk si můžeme ukládat více informací, jako např. záznam o překážce nebo informaci o tom, zda už byla daná oblast prozkoumána a budeme mít tak přehled o oblastech, resp. buňkách, které musíme navštívit, aby mohlo dojít k mapování jejich okolí.

Pro tuto reprezentaci mapy se hodí dvourozměrné pole, kde každá buňka bude zabírat určitou plochu prostoru. Pokud si určíme velikost buňky 1 cm x 1 cm, pak například buňka s indexy 5 a 3 může zabírat prostor mezi 5 cm a 6 cm pro souřadnici x a mezi 3 cm a 4 cm pro souřadnici y. Problém ovšem nastává při rozšiřování mapy, resp. ve zaznamenávání hodnot mimo rozsah dvourozměrného pole. Pro rozšiřování mapy v kladném směru obou os, bychom mohli použít některou z dynamických kolekcí, jako např. ArrayList<>. Při rozšiřování do záporného směru bychom, ale museli přepočítávat pole nebo kolekci.

Výhodnější strukturou je kvadrantový strom [27], kde v každém listu bude menší dvourozměrné pole. Strom můžeme rozšiřovat do všech směrů a další výhodou je i rychlé nalezení listu, které obsahuje pole, do kterého budeme zaznamenávat informace o buňkách. Složitost pro vyhledání listu je určena jako $\mathcal{O}(h)$, kde h je hloubka stromu, která je pro úplný kvadrantový strom s n listy dána jako $\log_4 n$ a výsledná složitost je tedy $\mathcal{O}\left(\log_4 n\right)$ [28]. Ukázka této struktury je zobrazena na obrázku 33, kde jsou modrou barvou znázorněna dvourozměrná pole, do kterých budeme ukládat informace.



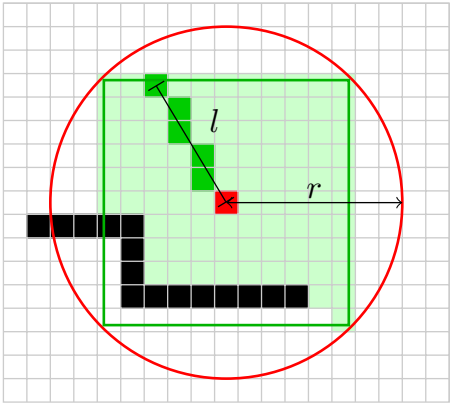
Obrázek 33: Struktura kvadrantového stromu

8.2 Viditelná oblast

Při autonomním mapování musíme vědět, která místa robot ještě neprozkoumal. V předem určené oblasti pro zmapování si tedy musíme pamatovat místa, která robot navštívil a ta místa, která robot nenavštívil, budeme následně mapovat. Za místa, která robot navštívil, nebudeme považovat pouze jednotlivé polohy robota, ale oblasti, které z daných poloh robot naskenoval.

Protože robot na určité pozici vidí místa, která jsou od něj vzdálená do vzdálenosti r, což je maximální dosah skeneru nebo vidí do vzdálenosti nejbližší překážky, tak si buňky v mapě pro tuto oblast budeme označovat za prozkoumané. Abychom nemuseli počítat body na kružnici s poloměrem r, resp. indexy buněk, které leží na této kružnici (červená kružnice na obrázku 34), tak za viditelnou oblast můžeme určit vepsaný čtverec do této kružnice (zelený čtverec). Buňky na obvodu vepsaného čtverce získáme jednoduše pomocí přičítání, resp. odečítání indexů od polohy robota (červená buňka). Následně provedeme rasterizaci úsečky l, což je úsečka mezi středem buňky s polohou robota a středem buňky na obvodu čtverce. Rasterizaci provádíme ve směru od polohy robota k okraji čtverce, a pokud při postupném „vybarvování" jednotlivých buněk pod úsečkou narazíme na překážku (černé buňky), tak rasterizaci aktuální úsečky l ukončíme a pokračujeme rasterizací další úsečky pro následující buňku na okraji čtverce.

Výsledné označení viditelné oblasti pak vidíme na obrázku 34 v podobě světle zeleně vybarvených buněk, kolem polohy robota. Při pohledu na spodní část viditelné oblasti je zřejmé, že nedošlo k označení buněk za překážkou (černé buňky).



Obrázek 34: Určení viditelné oblasti

8.2.1 Rasterizace úsečky pomocí Bresenhamova algoritmu

Pro rasterizaci úsečky můžeme použít Bresenhamův algoritmus [29]. Mějme úsečku definovanou dvěma body se souřadnicemi x_0, y_0 a x_1, y_1. Základní rovnice pro přímku definovanou pomocí těchto dvou bodů je [30]:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}.$$

Pro výpočet souřadnice y, pak:

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0 = \frac{\Delta y}{\Delta x}(x - x_0) + y_0,$$

kde $\frac{\Delta y}{\Delta x}$ určuje sklon úsečky. Ten si označme jako d.

V základním algoritmu předpokládejme, že máme nevertikální úsečku, která směřuje zleva doprava a zespodu nahoru a tedy platí $x_0 < x_1 \wedge y_0 < y_1 \wedge \Delta x \neq 0$. Algoritmus iteruje přes souřadnici x, začínající na hodnotě x_0 a končící na x_1. Při každé inkrementaci souřadnice x přičítáme sklon úsečky d k proměnné err, která začíná na nule. Po překročení hodnoty $0,5$, pak inkrementujeme souřadnici y o jedna a hodnotu err o jedna snížíme. V jazyce Java by pak funkce vypadala jako ve výpisu kódu 3.

Výpis 3: Základní Bresenhamův algoritmus

```
1   public void bresenhamLine(int x0, int y0, int x1, int y1) {
2       int deltaX = x1 − x0;
3       int deltaY = y1 − y0;
4       double d = deltaY / deltaX;
5       double err = 0;
6
7       int y = y0;
8       for (int x = x0; x <= x1; x++) {
9           System.out.println(x + ", " + y);
10          err += d;
11          if (err >= 0.5) {
12              y++;
13              err−−;
14          }
15      }
16  }
```

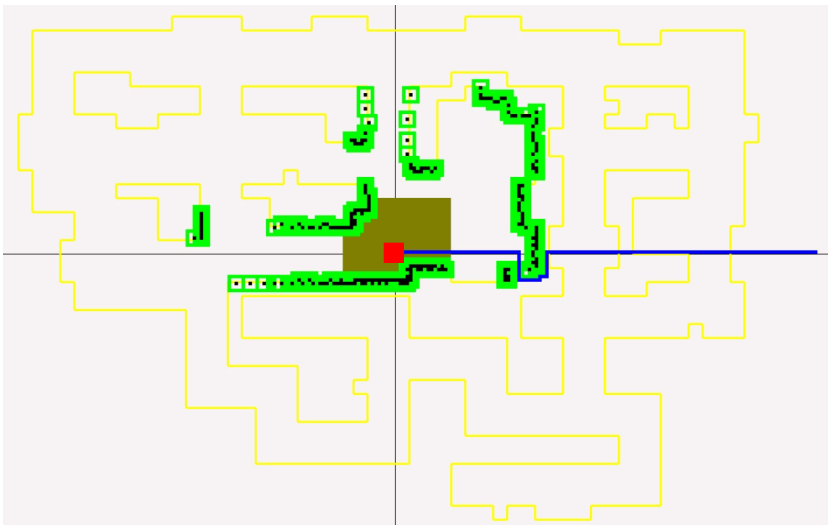
Nevýhodou tohoto přístupu je práce s desetinnými čísly u proměnných d a err. Těch se můžeme zbavit, pokud tyto proměnné (včetně konstanty $0,5$) vynásobíme hodnotou Δx. Dále je výhodné změnit způsob počítání hodnoty err, kdy nebudeme začínat na nule a končit na hodnotě $0,5$, resp. $0,5\,\Delta x$, ale budeme proměnnou err inicializovat na hodnotě $\Delta x/2$ a podmínkou pro inkrementaci souřadnice y bude překročení hodnoty nula.

Výsledný algoritmus, který obsahuje tyto změny, je uveden v příloze A. Zároveň má přidané další změny, které řeší použití pro jakýkoli směr úsečky. Díky tomu se dá použít pro vytvoření viditelné oblasti pro danou polohu robota.

8.3 Mapování okraje mapované oblasti

Při autonomním mapování jde o prozkoumání celé mapované oblasti, což v našem případě znamená, že každé místo v mapě, resp. každá buňka v mřížce získá příznak „prozkoumána", podle kapitoly 8.2. Zároveň buňky s překážkami, které jsou v průběhu skenování zaznamenány a místa, která nemůžeme vůbec prozkoumat, protože jsou celá obklopená překážkami, získají příznak „neprozkoumatelná". Jedním ze způsobů, jak takové mapování provést, je rozdělení mapování na dvě části. V první části zmapujeme vnější okraj mapy, při kterém už robot projede velkou část vnitřních oblastí mapovaného prostředí a po uzavření vnější části domapujeme vnitřní neprozkoumaná místa.

Pro zmapování vnějšího okraje mapy stačí, když si cíl pro tuto první část mapování určíme kousek za mapovanou oblastí. Algoritmus pro hledání cesty (viz kapitola 9) najde nejprve přímou cestu k tomuto cíli a při postupném objevování překážek bude vypočítanou cestu měnit. Na obrázku 35 je zobrazeno prostředí (žluté čáry), které robot (červený čtverec) zatím nezmapoval. Při prvním skenování si určil cíl za okrajem mapované oblasti a k tomuto cíli vytvořil cestu (modrá čára). Cesta zatím vede i přes překážky v prostředí, protože o nich robot zatím neví.



Obrázek 35: Určení cíle pro mapování okraje mapované oblasti

Po čase dojde i k zaznamenání první překážky na okraji mapované oblasti, která dělí počáteční polohu robota a cíl přímou cestou. Algoritmus si ale najde novou cestu kolem této překážky. Prozkoumávaná oblast se uzavře, když nebude existovat žádná cesta mezi robotem a cílem, který je určený mimo oblast pro mapování. Důvodem pro nenalezení cesty jsou buď překážky mezi robotem a cílem, nebo námi určený okraj mapované oblasti.

8.4 Mapování vnitřní oblasti

Po tom, co máme zmapovanou vnější oblast, která je uzavřená díky překážkám v prostředí nebo díky hranici mapované oblasti, můžeme domapovat zbylé vnitřní oblasti. Stačí, když začneme procházet jednotlivé buňky v mapě a když narazíme na první neprozkoumanou oblast (buňku), tak tuto buňku budeme považovat za aktuální cíl. Najdeme si k ní cestu (pomocí algoritmu, popsaného v kapitole 9) a když dojedeme s robotem blízko ní, tak nám ji algoritmus pro určení viditelné oblasti (viz kapitola 8.2) označí za prozkoumanou. Poté si najdeme další neprozkoumanou buňku a algoritmus opakujeme.

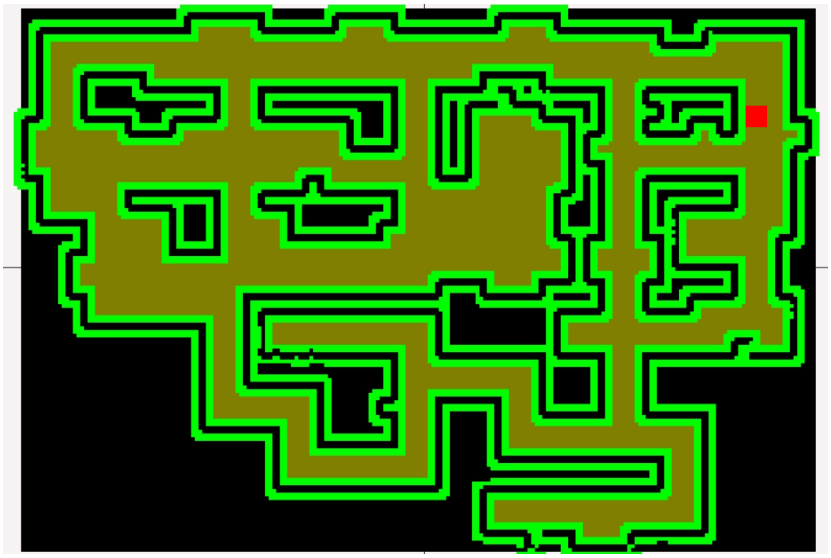
Při tomto přístupu však dochází k neefektivním jízdám robota, protože často nastává situace, kdy první neprozkoumaná buňka je např. v levé části mapy, následující buňka je v pravé části mapy, jako další neprozkoumaná buňka by byla označena opět taková, která je v levé části mapy. Tím by robot zbytečně projížděl prostředím z jednoho konce do druhého a zase zpět.

Efektivnější způsob tkví v nalezení shluků neprozkoumaných buněk. Když při postupném procházení mapy narazíme na neprozkoumanou buňku, tak si tuto buňku označíme za

cíl a také budeme hledat další neprozkoumané buňky v přímém okolí tohoto cíle. Získáme tím několik dalších cílů pro prozkoumání, které budou mít větší prioritu, než neprozkoumané buňky, nalezené postupným prohledáváním mapy. Na obrázku 36 vidíme, jak si robot našel cestu k prvnímu neprozkoumanému shluku buněk (prázdné místo v mapě) a pro zmapování tohoto shluku bude pokračovat na další nezmapovaný shluk buněk.

Obrázek 36: Mapování vnitřní oblasti

Takto robot projede všechny vnitřní oblasti, které v průběhu mapování okraje mapované oblasti nebyly prozkoumány. Zároveň také určí, která místa jsou neprozkoumatelná (černá místa na obrázku 36 a 37), díky tomu, že k nim algoritmus pro hledání cesty nenašel žádný způsob, jak by je mohl navštívit. Po dokončení mapování vnitřní oblasti máme tedy zmapovanou celou oblast, kterou jsme si na začátku určili. Ta je zobrazena na obrázku 37.



Obrázek 37: Výsledek autonomního mapování

9 Plánování cesty

Součástí autonomního řízení vozidel, konkrétně autonomního mapování, je určování cílů, které má robot navštívit. Efektivní výběr cílů jsme si již ukázali v předchozí kapitole 8. V ní jsme znali polohu robota, bod A a určili jsme si cíl, bod B, který chceme prozkoumat. K dopravení robota do tohoto cíle se ještě musí vypočítat cesta mezi těmito body v mapě. Půjde tedy o nalezení sekvence buněk v mapě, kterou jsme definovali v podkapitole 8.1.

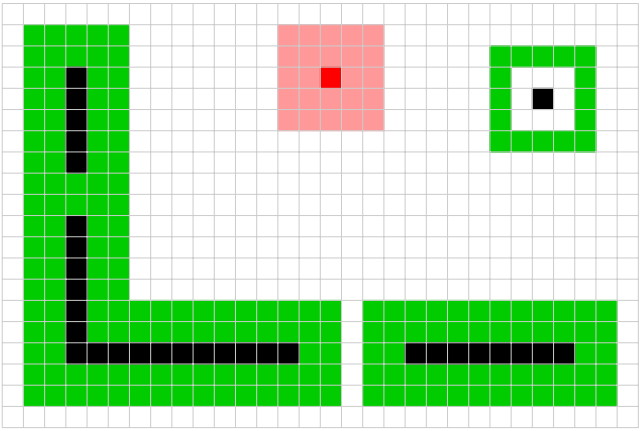
Tato cesta musí splňovat několik kritérií, aby se dala správně a efektivně použít pro řízení robota. Hlavním kritériem je umístění cesty pouze do volných buněk a zároveň cesta nesmí vést přes překážku. Nalezená cesta ovšem nesmí vést ani mezi dvěma překážkami, mezi kterými robot nemůže projet. Také by se mělo jednat o cestu, jejíž délka se blíží nejkratší možné cestě mezi body A a B, ale nemusí to být přímo nejkratší cesta. Poslední kritérium se týká bezpečnosti jízdy a tak by se měla vypočítat bezpečnější cesta, která je více vzdálená od překážek, aby nedošlo ke srážce robota s překážkou. A právě hledání takové cesty, která splňuje všechna tato kritéria, si ukážeme v této kapitole.

9.1 Rozšířené zdi

Předpokládejme, že máme vytvořenou část mapy, kterou robot pomocí skenování vytvořil. Do určitých buněk přidal záznam o překážce a v takto vytvořené mapě chceme najít cestu, která vede z polohy robota do cíle mezi těmito překážkami. Jednou z věcí, které musíme zajistit, je zjištění, zda může robot mezi překážkami projet.

Pokud by robot zabíral pouze velikost jedné buňky, tak bychom nemuseli v této části nic řešit a pokud by mezi překážkami existovala jakkoli úzká cesta, třeba jen jedno volné políčko, tak by mohl robot mezi těmito překážkami projet. Reálný robot ovšem zabírá více, než jednu buňku a tak cesta, kterou bychom našli výše popsaným způsobem, by byla pro robota zbytečná, protože by jí nemohl projet. Stejný problém nastává i pro cestu těsně okolo zdí, která je vždy nejkratší. V tomto případě bychom museli testovat, jak blízko zdi může být výsledná cesta, aby ji robot mohl použít.

Na tento problém můžeme pohlížet i z opačného hlediska. Nebudeme za robota považovat více buněk, ale jen jednu a každou naskenovanou překážku obklopíme „rozšířenou zdí". Po této úpravě můžeme hledat cestu v jakékoli volné buňce a máme zaručeno, že robot dokáže v těchto místech projet, pokud velikost rozšířené zdi bude větší, než polovina velikosti robota. Výsledek vidíme na obrázku 38, kde jsme původní velikost robota (světle červená barva) zmenšili na jednu buňku (tmavě červená barva) a k překážkám (černé barvy) jsme přidali rozšíření o velikosti dvou buněk (zelené buňky). Při reálném použití je lepší zvolit o málo větší velikost rozšířené zdi, abychom měli větší jistotu, že nedojde ke srážce robota s překážkou. Účinek je patrný v levé části obrázku, kde je mezera mezi překážkami úzká pro průjezd robota a tak došlo k jejímu zacelení. Druhá mezera mezi překážkami ve spodní části obrázku je ovšem dostatečně široká, a pokud robot pojede v jejím středu, což je jediná volná buňka, tak může touto mezerou projet. Pokud umístíme robota, resp. jeho střed do jakékoli volné buňky, tak máme jistotu, že se bude celý robot nacházet ve volném prostředí a nebude například z poloviny v překážce.



Obrázek 38: Vytvoření rozšířených zdí

9.2 Vyhledání nejkratší cesty algoritmem A*

Pro vyhledání cesty mezi polohou robota a cílem použijeme algoritmus A* [31], který vychází z Dijkstrova algoritmu, ke kterému přidává heuristickou funkci $h(x)$. A* prochází graf, což je v našem případě mapa buněk, kde každá buňka představuje uzel, a hrany v tomto grafu jsou vytvořeny mezi sousedními buňkami, které mají společnou stranu. Ohodnocení každé hrany je 1 a udává tedy vzdálenost mezi buňkami.

Algoritmus začíná v uzlu A a postupně prochází dalšími sousedními uzly, kde každému z nich určí hodnotu podle funkce $f(x)$, která je definována jako:

$$f(x) = g(x) + h(x),$$

kde $g(x)$ je funkce, která určuje vzdálenost mezi počátečním uzlem A a aktuálně zpracovávaným uzlem x. Pro bod A je tedy $g(A) = 0$ a pro jeho sousední uzly je $g(x) = 1$. Funkce $h(x)$ je výše zmíněná heuristická funkce a určuje nám, zda postupujeme optimálním směrem k cíli. Pro heuristickou funkci $h(x)$ se používá vzdálenost mezi aktuálním uzlem a cílem B, pokud hledáme nejkratší cestu.

$$h(x) = distance(x, B)$$

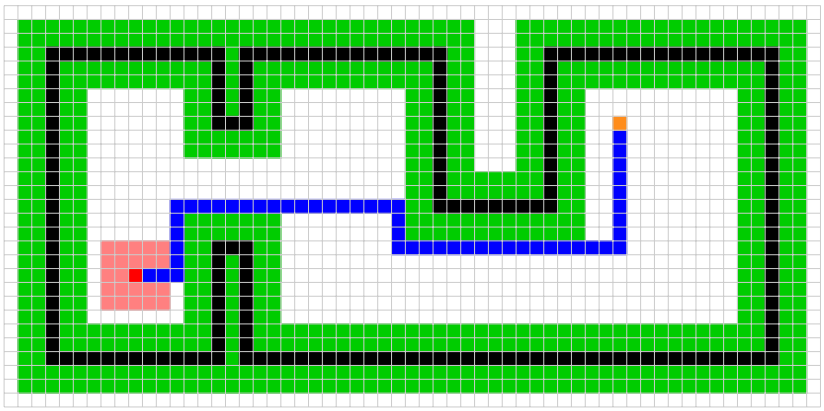
Obvykle se určuje jako vzdušná vzdálenost mezi těmito uzly. V našem případě můžeme zjistit přímou nejkratší vzdálenost mezi dvěma buňkami jako součet rozdílů jejich indexů x a y.

$$cellDistance(x_0, y_0, x_1, y_1) = |x_0 - x_1| + |y_0 - y_1|$$

Až dojde k ohodnocení sousedních uzlů, tak se aktuálně zpracovávaný uzel označí za „uzavřený" a už se nebude zpracovávat. Sousední uzly zatím nejsou zpracované, ale pouze ohodnocené a tak jsou přidány do kolekce „otevřených uzlů". Pro zpracování dalšího uzlu si z kolekce „otevřených uzlů" vybereme takový, který byl ohodnocen nejmenší hodnotou funkce $f(x)$.

Algoritmus končí, pokud začneme zpracovávat cílový uzel B. Výsledné uzly, resp. buňky, které tvoří nejkratší cestu, pak najdeme, když budeme procházet graf od cílového uzlu B k počátečnímu uzlu A. Pro nalezení dalšího uzlu cesty pouze vybereme takový sousední uzel, který má nejnižší hodnotu $f(x)$.

Na obrázku 39 je zobrazena cesta (modré barvy), kterou algoritmus A* našel mezi počáteční polohou robota (červená barva) a cílem (oranžové barvy). Vidíme, že se jedná o nejkratší cestu mezi těmito buňkami.



Obrázek 39: Nalezení nejkratší cesty algoritmem A*

9.3 Změna heuristiky pro vyhledání bezpečnější cesty

Při použití heuristické funkce, která měří vzdálenost mezi buňkami, hledá algoritmus A* nejkratší cestu. Taková cesta je vždy vedena z velké části podél zdí, což je i patrné z obrázku 39. Při menší chybě lokalizace by se tak mohlo stát, že se robot ocitne ve zdi nebo v rozšířené zdi, protože je jeho poloha velmi blízko zdi.

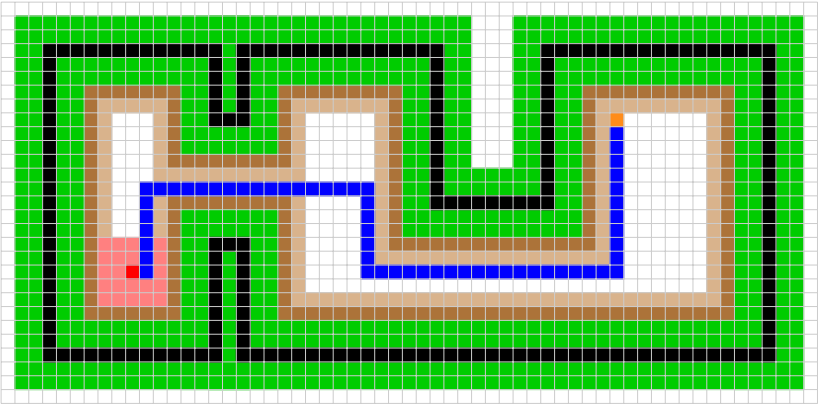
Vzhledem k tomu, že nepotřebujeme, aby robot používal nejkratší cestu v rámci buněk v mapě, ale spokojíme se i s cestou, která je nejkratší pouze v rámci uliček v mapě, tak můžeme tuto heuristickou funkci modifikovat. Mnohem bezpečnější způsob pro pohyb robota je ve větší vzdálenosti od zdi a proto budeme v heuristické funkci znehodnocovat buňky, které jsou příliš blízko zdi. Čím blíže bude buňka u zdi, tím vyšší výslednou hodnotu $h(x)$ bude mít. Zároveň si můžeme stanovit určitou maximální vzdálenost od zdi w_{max}, která nás zajímá a pokud budeme zpracovávat buňku, která je dále, tak budeme používat pouze výše zmíněnou heuristickou funkci. Výsledná heuristická funkce tedy může vypadat jako:

$$h(x) = \begin{cases} \| x - B \| & \text{pro } w(x) > w_{max} \\ \| x - B \| + c\,(w_{max} - w(x)) & \text{pro } w(x) \leq w_{max} \end{cases},$$

kde funkce $w(x)$ určuje vzdálenost k nejbližší překážce a c je konstanta, která určuje, jak moc vzdálenost od zdi ovlivňuje výslednou heuristiku.

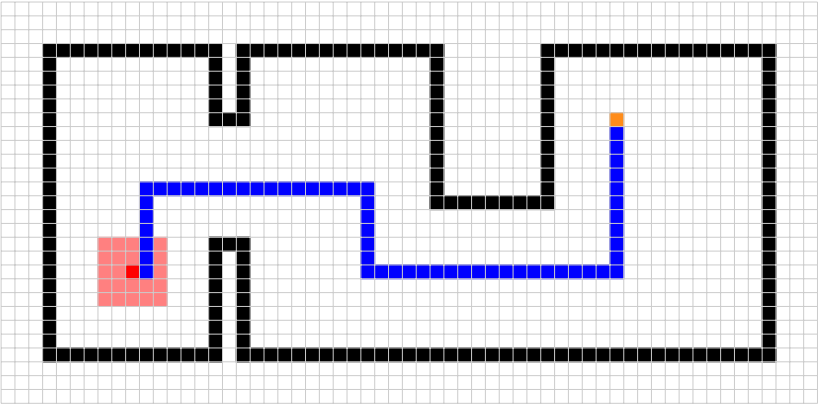
Pokud takto změníme heuristickou funkci, tak nám bude algoritmus A* hledat cestu, která je více vzdálená od zdí. Nejlépe bude hledat cestu, která je vzdálená od zdi o w_{max}, pokud zvolíme dostatečně vysoké c. Současně však najde i cestu mezi úzkými překážkami, pokud mezi nimi dokáže robot projet a jako nejbezpečnější místo průjezdu určí střed mezi nimi. Na obrázku 40 vidíme cestu (modrá barva), která byla nalezena pomocí takto změněné heuristické funkce. Robot (červený čtverec) projíždí mezi zdmi (černé barvy), které jsou obklopeny rozšířenými zdmi (zelené barvy). A také, pokud může, tak se drží dál od zdí.

Hodnoty funkce $w(x)$ jsou na obrázku zakresleny hnědou barvou. Čím je hnědá barva tmavší, tím je hodnota $w(x)$ nižší a proto více ovlivňuje heuristickou funkci.



Obrázek 40: Nalezená cesta po změně heuristiky

Když si zobrazíme obrázek 40 bez rozšířených zdí a heuristické funkce, tak vidíme, že robot volí cestu, která je blízká té nejkratší, je bezpečná a splňuje i další kritéria, která jsme uvedli na začátku této kapitoly. Zobrazení robota, cesty k cíli a překážek je ukázáno na obrázku 41.



Obrázek 41: Zobrazení výsledné bezpečné krátké cesty

10 Autonomní parkování

V předchozích kapitolách jsme se zabývali technikami a algoritmy, které potřebujeme pro uskutečnění stanoveného cíle. V této kapitole si ukážeme, jak můžeme využít výše popsané znalosti pro zaparkování robota do parkovacího místa, které je umístěno v neznámém prostředí. Nejprve však musíme definovat, co považujeme za parkovací místo. Následně popíšeme jednotlivé kroky, které je nutné provést, aby se robot dokázal pohybovat v neznámém prostředí, postupně ho prozkoumával, vyhýbal se překážkám a po nalezení vhodného parkovacího místa, do tohoto místa zajel. Nakonec si robot po pár vteřinách čekání najde cestu zpět k místu, ze kterého startoval a dojede zpět na počáteční polohu.

10.1 Nalezení parkovacího místa

Definování parkovacího místa závisí na robotovi, pro kterého je určeno. Pro testování této práce používáme robota Neato XV-15, který má rozměry šířky a délky 32,5 x 30,5 cm (viz kapitola 3). K parkovacímu místu je vhodné přidat určitou toleranci na každou stranu robota z důvodu nepřesnosti skeneru a pohybového systému. Když přidáme necelých 15 cm, tak můžeme parkovací místo definovat jako čtverec o délce strany 60 cm.

Cílem je tedy nalézt takto velký čtverec, jehož rohy jsou význačnými body a představují rohy mezi stěnami v naskenovaném prostředí. Z důvodů chyb měření a pohybu je velmi malá šance, že takový čtverec najdeme. Budeme raději hledat takovou oblast, která má délky stran mezi 50 - 70 cm a úhly mezi těmito stranami nemusí být kolmé, ale stačí, když budou svírat úhel mezi 85° a 95°. Přidali jsme tak toleranci $\pm$ 10 cm pro délku strany a $\pm$ 5° pro úhel mezi stranami.

Pro výběr čtyř bodů z význačných bodů, které tvoří oblast parkovacího místa, nestačí postupně tyto body procházet a hledat sekvenci, která by splňovala výše uvedené podmínky pro parkovací oblast, protože neznáme pořadí význačných bodů. Musíme proto postupovat jinak. Každému význačnému bodu L_i přiřadíme takové body, které jsou k němu vzdálené nejméně $dist_{min}$, což je 50 cm a nejvíce $dist_{max}$, 70 cm. Tím získáme páry význačných bodů a některé z nich mohou tvořit stěny parkovacího místa. Nyní potřebujeme zjistit, které stěny svírají větší úhel než $\beta_{min} = 85°$ a menší úhel než $\beta_{max} = 95°$. Zároveň tyto stěny musí mít jeden společný bod, u kterého se vnitřní úhel měří. Pro toto zjištění budeme provádět následující operace při procházení páru:

1. Aktuálně zpracovávaným význačným bodem v páru je bod L_0, který má přiřazené body L_3, L_6 a L_9. Do pomocného pole $buffer$ si uložíme bod L_0.
2. Ve vnitřním cyklu začneme procházet přiřazené body L_3, L_6 a L_9.
   (a) Ve vnitřním cyklu zpracováváme bod L_3 a přidáme jej tedy do pole $buffer$.
   (b) Zjistíme si, jaké body má v páru bod L_3, těmi jsou body L_4 a L_6.
   (c) V dalším vnořeném cyklu budeme tyto dva body procházet.
      i. Budeme zpracovávat bod L_4.
      ii. Pokud vnitřní úhel mezi body L_0, L_3 a L_4 není mezi β_{min} a β_{max}, tak budeme zpracovávat další bod L_6. Pokud úhel odpovídá, tak přidáme bod L_4 do pole $buffer$ a budeme procházet body, které jsou v páru s bodem L_4, tím je pouze bod L_2.

A. V tomto cyklu budeme zpracovávat bod L_2.
B. Jestli je vnitřní úhel mezi body L_3, L_4 a L_2 takový, jaký hledáme, tak jsme našli parkovací místo s body L_0, L_3, L_4, L_2 a můžeme si tyto body uložit, pokud jsme takové místo ještě nenašli.

Výsledkem algoritmu je seznam čtveřic bodů, kde každá čtveřice odpovídá parkovacímu místu. K určení cíle pro výpočet cesty můžeme určit bod, který je uprostřed daného místa. Pro parkovací oblast s body $L_0 = [x_0, y_0]$, $L_1 = [x_1, y_1]$, $L_2 = [x_2, y_2]$ a $L_3 = [x_3, y_3]$ můžeme vypočítat střed této oblasti jako centroid $C = [x_c, y_c]$ polygonu, tvořeného těmito body. Výpočet jeho souřadnic vypadá následovně:

$$x_c = \frac{\sum_{i=0}^{3} x_i}{4}, \quad y_c = \frac{\sum_{i=0}^{3} y_i}{4}.$$

10.2 Popis kroků pro zaparkování

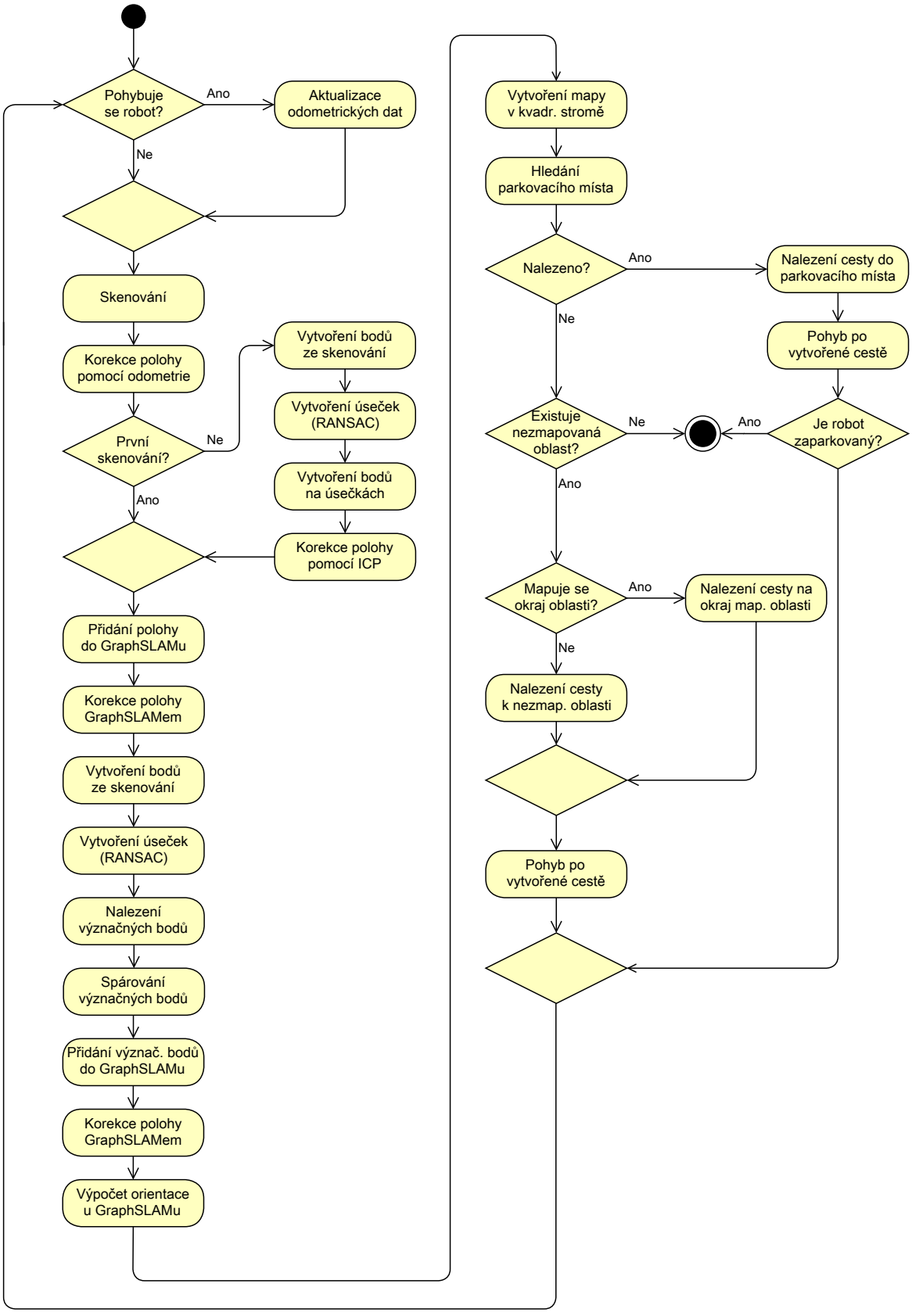
V této podkapitole si ukážeme kroky, které musí robot vykonat, aby dokázal pomocí LIDARového skeneru najít vhodné místo na zaparkování a zaparkoval do něj. Diagram aktivit, který popisuje činnosti robota od začátku, až po zaparkování je zobrazen na obrázku 42. Návrat na počáteční pozici z parkovacího místa je pak velmi podobný a popíšeme jej v této podkapitole pouze slovně.

Robot začíná zpracováním odometrických dat, pokud vykonává pohyb. Tím přibližně zjistí, o kolik se v mapě posunul a otočil. Tyto hodnoty (Δx, Δy a $\Delta\alpha$) vypočítáme podle kapitoly 4. Následně dojde ke skenování (pomocí příkazů z podkapitoly 3.2.3), které je vhodné spustit po zjištění odometrických dat, protože dochází ke zpoždění kvůli přenosu naskenovaných dat z robota. Toto zpoždění by mělo za následek, nepřesné určení odometrických hodnot pro naskenovaná data. Díky odometrickým datům můžeme provést první korekci polohy robota (viz kapitola 7.1). Pokud se nejedná o první iteraci a máme uloženy body z předchozího skenování, tak můžeme provést korekci pomocí spárování naskenovaných snímků algoritmem ICP (kapitola 7.2). Tomu však musí předcházet zpracování LIDARových dat, které jsme si ukázali v kapitole 5. Polohu a orientaci robota máme nyní zpřesněnou díky odometrii a ICP, další zpřesnění můžeme provést, když přidáme tento odhad pohybu do matice pro GraphSLAM a vypočítáme tím pravděpodobnější polohu robota. Díky této přesnější poloze přepočítáme zjištěné body a úsečky z LIDARových dat a také si zjistíme význačné body pro aktuální naskenovaný snímek podle kapitoly 5.3. Význačné body doplníme do matic pro GraphSLAM a provedeme finální korekci polohy robota spolu s určením jeho orientace (blíže v podkapitolách 7.4 a 7.5).

Po určení nejpravděpodobnější polohy robota spolu s jeho orientací můžeme vytvořit mapu v kvadrantovém stromu. Protože GraphSLAM neprovádí pouze korekci aktuální polohy, ale i všech předchozích poloh, tak musíme vytvořit tuto mapu postupným přidáváním všech naskenovaných snímků, které jsou vůči sobě posunuty a otočeny na základě zpřesněných poloh a orientací robota. V této mapě budeme později hledat cestu. Nejprve musíme zjistit, zda v doposud prozkoumaném prostředí nachází parkovací místo. To je tvořeno z význačných bodů, jejichž nejpravděpodobnější polohu získáme opět z GraphSLAMu a samotné nalezení bodů, které tvoří parkovací místo, provedeme podle předchozí podkapitoly 10.1. Pokud parkovací místo najdeme, tak si k němu vytvoříme cestu v mapě algoritmem A* z kapitoly 9. Pokud jsme parkovací místo nenašli, tak budeme provádět autonomní mapování,

při kterém budeme vždy kontrolovat, zde jsme již neobjevili vhodné místo pro parkování. Nejprve se zaměříme na mapování vnější oblasti a nakonec budeme mapovat vnitřní části, které ještě nebyly zmapovány. Postup pro autonomní mapování je popsán v kapitole 8. Po těchto krocích se algoritmus vrací na začátek a provádí se tak dlouho, dokud robot nenajde parkovací místo a nezaparkuje do něj nebo dokud nebude zmapována celá oblast.

Po skončení této části se může robot vrátit na počáteční polohu (bod $[0, 0]$ v mapě). Stačí, když si do tohoto bodu vytvoří cestu algoritmem A* a bude provádět stejné kroky, které jsme popsali výše, kromě hledání parkovacího místa a nezmapovaných oblastí.



Obrázek 42: Diagram aktivit pro parkování

11 Závěr

Plně autonomní řízení vozidel je bezesporu zajímavá oblast, která, dle mého názoru, bude v budoucnu stále více uplatňovaná v běžném životě lidí. Představa, ve které pouze nastoupíme do vozidla a to nás samo a bezpečně zaveze k cíli naší cesty, je velmi lákavá. První ukázky toho, že můžeme vytvořit takováto vozidla, která jsou řízena bez účasti člověka, již dnes existují. Výzkum v této oblasti je ale ještě poměrně mladý a tak zatím nejsou vyřešeny všechny problémy, se kterými se může samostatné řízení setkat. Většina těchto problémů se týká lokalizace vozidla v prostoru. Problémem mohou být nedokonalosti skenerů, které se nemohou lokalizovat v nepříznivém počasí, kdy je například silnice zasněžená. V příznivém počasí ovšem skenery fungují dobře. Větší problém pak představuje zjištění změny polohy a orientace vozidla v čase, pomocí pohybového systému vozidla. Nejnovější přístup pro řešení těchto problémů využívá teorii pravděpodobnosti, ve které počítá s chybami, které mohou při pohybu a skenování nastat.

V této práci jsem se snažil předvést konkrétní a úplné řešení problémů současné lokalizace a mapování, spolu s uvedením matematických vztahů, ze kterých je řešení odvozeno. Popsány jsou jednotlivé kroky od získání, zpracování a analýzy dat z LIDARového skeneru, přes řešení samostatné lokalizace (kdy máme dostupnou mapu prostředí), až po řešení problému současné lokalizace a mapování, při kterém se robot pohybuje v neznámém prostředí. V další části jsem uvedl, jak můžeme robota naučit hledání cílů a bezpečných cest, aby prozkoumal celou oblast autonomně.

Klíčovým prvkem pro samostatnou lokalizaci nebo pro současnou lokalizaci a mapování je hledání význačných bodů ve skenovaných snímcích a propojení vzájemně korespondujících bodů v rámci několika naskenovaných snímků. Pro detekci význačných bodů jsem se zaměřil na rohy, které jsou tvořeny překážkami ve vnitřním prostředí. Navrhl jsem způsob, jak můžeme roh detekovat a jak jeho polohu zpřesnit. Pro spárování význačných bodů se nejčastěji používá metoda nejbližšího souseda, která potřebuje, aby byly korespondující body blízko sebe. Kvůli chybám pohybového systému však tento předpoklad nemusí být vždy splněn. Pro prostředí, které obsahuje více význačných bodů, jsem navrhl metodu korespondujících vzdáleností mezi význačnými body. Tato metoda nepotřebuje, aby byly korespondující body blízko sebe, a nezáleží tedy, k jaké chybě pohybového systému došlo.

Všechny tyto techniky a algoritmy se dají využít mnoha směry. Jedním z nich může být výše zmíněné cestování vozidlem, bez účasti člověka. Druhým může být například automatické parkování s vozidlem, ve kterém není posádka. A právě tento druhý úkol jsem otestoval s robotem Neato XV-15, kdy jsem jej umístil do neznámého prostředí, které obsahovalo překážky a parkovací místo. Nechal jsem robota prozkoumávat toto neznámé místo a po chvíli sám přijel k místu s parkovištěm. Toto místo správně vyhodnotil jako volné parkovací místo a sám do něj také zajel. Po zaparkování měl zmapovanou přibližně polovinu neznámé oblasti a po tomto úkolu měl robot dojet na místo, ze kterého vyjížděl. Cestu zpět si zvolil podle nejkratší bezpečné cesty, tu následoval a dojel tak k místu počáteční polohy, čímž splnil předem určený úkol.

12 Reference

[1] ROBOTS CONQUER DARPA Grand Challenge. In: *DARPA - Press Releases* [online]. 2005 [cit. 2013-04-02]. Dostupné z: <http://archive.darpa.mil/grandchallenge05/GC05winnerv2.pdf>.

[2] A HUGE LEAP FORWARD FOR ROBOTICS R&D. In: *DARPA - Press Releases* [online]. 2005 [cit. 2013-04-02]. Dostupné z: <http://archive.darpa.mil/grandchallenge05/GC05winnerFINALwTerraM.pdf>.

[3] DARPA Grand Challenge (2005). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-02]. Dostupné z: <http://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2005)>.

[4] Say Hello to Stanley. *Wired* [online]. 2006 [cit. 2013-04-02]. Dostupné z: <http://www.wired.com/wired/archive/14.01/stanley.html?pg=5>.

[5] Announcements. *Stanford Racing* [online]. 2006 [cit. 2013-04-02]. Dostupné z: <http://cs.stanford.edu/group/roadrunner/old/announcements.html>.

[6] DARPA Grand Challenge (2007). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-02]. Dostupné z: <http://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2007)>.

[7] Google driverless car. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-02]. Dostupné z: <http://en.wikipedia.org/wiki/Google_driverless_car>.

[8] CES 2013: Audi Demonstrates Autonomous Piloted Parking with A7 (Video). In: *Fourtitude* [online]. 2013 [cit. 2013-04-02]. Dostupné z: <http://fourtitude.com/news/Audi_News_1/ces-2013-audi-demonstrates-autonomous-piloted-parking/>.

[9] LIDAR. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-03]. Dostupné z: <http://en.wikipedia.org/wiki/LIDAR>.

[10] MAŇÁK, Roman. Rayleighův a Mieův rozptyl I. In: *Parhelium* [online]. 2007 [cit. 2013-04-04]. 3/2007. Dostupné z: <http://ukazy.astro.cz/gal/Parhelium200703.pdf>.

[11] Ramanova spektroskopie. MATĚJKA, Pavel. *Návody pro laboratorní cvičení z analytické chemie III*. Praha: VŠCHT, 2002. ISBN 80-7080-466-1.

[12] Laser rangefinder. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-06]. Dostupné z: <http://en.wikipedia.org/wiki/Laser_rangefinder>.

[13] MCKINION, J.M., J.L. WILLERS a J.N. JENKINS. Comparing high density LIDAR and medium resolution GPS generated elevation data for predicting yield stability. *Computers and Electronics in Agriculture*. 2010, roč. 74, č. 2, s. 244-249. ISSN 01681699.

DOI: 10.1016/j.compag.2010.08.011. Dostupné z: <http://linkinghub.elsevier.com/retrieve/pii/S0168169910001559>.

[14] NEATO ROBOTICS, Inc. *Neato Vacuum User's Guide*. Mountain View, 2010.

[15] Programmer's Manual: Response Syntax. *Neato Robotics* [online]. 2013 [cit. 2013-04-07]. Dostupné z: <http://www.neatorobotics.com/programmers-manual/response-syntax/>.

[16] Odometrie. In: *Robotika* [online]. 2005 [cit. 2013-04-14]. Dostupné z: <http://robotika.cz/guide/odometry/>.

[17] FISCHLER, Martin A. a Robert C. BOLLES. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*. 1981, roč. 24, č. 6, s. 381-395. ISSN 00010782. DOI: 10.1145/358669.358692. Dostupné z: <http://portal.acm.org/citation.cfm?doid=358669.358692>.

[18] Lineární regrese. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-22]. Dostupné z: <http://cs.wikipedia.org/wiki/Line%C3%A1rn%C3%AD_regrese>.

[19] DELLAERT, Frank, Dieter FOX, Wolfram BURGARD a Sebastian THRUN. Monte Carlo Localization for Mobile Robots. In: *IEEE International Conference on Robotics and Automation (ICRA99)* [online]. 1999 [cit. 2013-04-30]. Dostupné z: <http://www.ri.cmu.edu/publication_view.html?pub_id=533>.

[20] LITSCHMANNOVÁ, Martina. *Vybrané kapitoly z pravděpodobnosti* [online]. Ostrava, 2011 [cit. 2013-04-10]. Dostupné z: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/vybrane_kapitoly_pravdepodobnost.pdf>.

[21] THRUN, Sebastian. Particle Filters in Robotics. In: *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)* [online]. 2002 [cit. 2013-04-30]. Dostupné z: <http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf>.

[22] Simultaneous localization and mapping. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-14]. Dostupné z: <http://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping>.

[23] RUSINKIEWICZ, Szymon a Marc LEVOY. Efficient variants of the ICP algorithm. In: *3-D Digital Imaging and Modeling* [online]. 2001, s. 145-152 [cit. 2013-04-23]. Dostupné z: <http://graphics.stanford.edu/papers/fasticp/fasticp_paper.pdf>.

[24] EGGERT, David Wayne, A. LORUSSO a Robert Burns FISHER. Estimating 3-D rigid body transformations: a comparison of four major algorithms. In: *Machine Vision and Applications: Special issue on performance evaluation* [online]. 1997 [cit. 2013-04-23]. DOI: 10.1007/s001380050048. Dostupné z: <http://www.cs.duke.edu/researchers/artificial_intelligence/temp/eggert_rigid_body_transformations.pdf>.

[25] THRUN, Sebastian a Michael MONTEMERLO. The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. In: *The International Journal of Robotics Research* [online]. 2006 [cit. 2013-04-25]. DOI: 10.1177/0278364906065387. Dostupné z: <http://robots.stanford.edu/papers/thrun.graphslam.pdf>.

[26] Mean of circular quantities. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-26]. Dostupné z: <http://en.wikipedia.org/wiki/Mean_of_circular_quantities>.

[27] FINKEL, Raphael A. a Jon Louis BENTLEY. Quad trees a data structure for retrieval on composite keys. In: *Acta informatica*. Berlin: Springer-Verlag, 1974, s. 1-9. ISSN 0001-5903. DOI: 10.1007/BF00288933. Dostupné z: <http://dx.doi.org/10.1007/BF00288933>.

[28] DVORSKÝ, Jiří. *Algoritmy I.* [online]. 2007 [cit. 2013-04-27]. Dostupné z: <http://www.cs.vsb.cz/dvorsky/Download/SkriptaAlgoritmy/Algoritmy.pdf>.

[29] BRESENHAM, Jack Elton. Algorithm for computer control of a digital plotter. In: *IBM Systems Journal*. s. 25-30. ISSN 0018-8670. DOI: 10.1147/sj.41.0025.

[30] Bresenham's line algorithm. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-28]. Dostupné z: <http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm>.

[31] HART, Peter E., Nils J. NILSSON a Bertram RAPHAEL. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *SIGART Bulletin* [online]. New York, NY: Special Interest Group on Artificial Intelligence, Association for Computing Machinery, 1972 [cit. 2013-04-30]. ISSN 0163-5719. DOI: 10.1145/1056777.1056779. Dostupné z: <http://doi.acm.org/10.1145/1056777.1056779>.

A Bresenhamův algoritmus

```
1  public void bresenhamLine(int x0, int y0, int x1, int y1) {
2      int deltaX = Math.abs(x1 − x0);
3      int deltaY = Math.abs(y1 − y0);
4
5      int stepX = x0 < x1 ? 1 : −1;
6      int stepY = y0 < y1 ? 1 : −1;
7
8      int error = (deltaX > deltaY ? deltaX : −deltaY) / 2;
9
10     while (true) {
11         System.out.println(x + ", " + y);
12
13         if (x0 == x1 && y0 == y1) {
14             break;
15         }
16
17         int error2 = error;
18
19         if (error2 > −deltaX) {
20             error −= deltaY;
21             x0 += stepX;
22         }
23
24         if (error2 < deltaY) {
25             error += deltaX;
26             y0 += stepY;
27         }
28     }
29 }
```